# Memory

## Modeling defaults in a loans for potential clientes

### Objectvive and general view

This is comes as a kaggle challenge from American Express. this challenge aim to predict the potential clients defaults on loans with 190 columns of data divided into 5 categories:

- D_* = Delinquency variables
- S_* = Spend variables
- P_* = Payment variables
- B_* = Balance variables
- R_* = Risk variables
- target in binary, 1 being default

the data for each client had diferent dimentions, depending on the months of data that each client had, ranging from 1 month to 13 months. The 13 months data being the vast majority of the data. In this project I will focus on data that has a dimention between 1 and 7 months due to limiting time of the deliverable. To achive this I first had to download the data and make it readable, considering that the training data was originaly 16gb, my computer could not handle reading a DataFrame that size so this code need to be implemented:

### Download data

here I change the type in case f numerical values being type str

```
def change_type(df,n=0):
    col=df.columns[n]

    if df[col].dtype=="O":
            try:
               df=df.astype({col:"float64"})

            except:
                df=df.astype({col:"str"})

    if n==len(df.columns)-1:
        return df
    else:
        n+=1
        return change_type(df,n)
```

here I create 2 columns and define all time columns

```
def clean_col(df):

    df["S_2"]=df["S_2"].astype('datetime64[ns]')
    df["Year"]=df["S_2"].dt.year
    df["Month"]=df["S_2"].dt.month

    return df
```

here I make a list of the permutation of each dataframe in the list of daframes, so that all values for each client will be in the same data frame

```
def reage_list(long,n=0,new_lis1=[],new_lis2=[]):

    new_lis1=list(map(lambda x: [long[n],x],long))

    new_lis2=new_lis2+new_lis1

    if n==len(long)-1:

        new_lis2=list(filter(lambda x:x  if x[0]!=x[1] else None,new_lis2))

        return new_lis2

    else:

        n+=1

        return reage_list(long,n,new_lis1,new_lis2)
```

here I concatonate all the customer_ID into the same data frame and drop that value from the prior dataframe

```
def organise_df(data,inter,n=0):
    now=inter[n]
    try:

        data[now[0][1]]=pd.concat([data[now[0][1]],data[now[0][0]][data[now[0][0]]
["customer_ID"]==now[1][0]]])

        data[now[0][0]]=data[now[0][0]].drop(data[now[0][0]][data[now[0][0]]
["customer_ID"]==now[1][0]].index)

    except:
        pass

    if n==len(inter)-1:
        return data
    else:
        n+=1
        return organise_df(data,inter,n)
```

and here I join them all so that I could read theses dataframes

```
def auto_sys(csv):

    chunks=pd.read_csv(csv,chunksize=500000)

    with cf.ThreadPoolExecutor() as executor:
        data=list(executor.map(lambda x:x , chunks))

    arr=np.arange(len(data)).tolist()

    arr_lis=reage_list(arr)

    # here I check using the permutation list the interseption that exist between dataframes in customer_ID
    with cf.ThreadPoolExecutor() as executor:
        inter=list(executor.map(lambda x:[x, np.intersect1d(data[x[0]]["customer_ID"].unique(), (data[x[1]]["customer_ID"].unique())).tolist()] ,arr_lis))


    data=organise_df(data,inter)



    with cf.ThreadPoolExecutor() as executor:
        data=list(executor.map(clean_col,data))



    return data
```

here I map out all the different month data that exists per Dataframe and had that to a dicionary where the key is the number of months

```
def map_data(data,gr_data,num_data,n=0,dic={}):



    array_ID=gr_data[gr_data["S_2"]==num_data[n]]["customer_ID"].unique()

    new_frame=data[data["customer_ID"].isin(array_ID)]

    dic.update({num_data[n]:new_frame})

    if n==len(num_data)-1:
        return dic
    else:
        n+=1
        return map_data(data,gr_data,num_data,n,dic)

```

here I make a dicionary to group data by month per customer_ID

```
def dic_maker(list_link):

    data=pd.read_csv(list_link)

    gr_data=data.groupby("customer_ID")["S_2"].count().to_frame().reset_index()

    num_data=list(gr_data["S_2"].unique())

    max_val=max(num_data)

    num_data.pop(num_data.index(max_val))

    n=0

    dic={}

    dic=map_data(data,gr_data,num_data,n,dic)

    array_ID=gr_data[gr_data["S_2"]==max_val]["customer_ID"].unique()

    data=data[data["customer_ID"].isin(array_ID)]

    dic.update({max_val:data})

    del data

    return dic
```

here I run the previouly made functions using threads

```
with cf.ThreadPoolExecutor() as excutor:
        data=list(excutor.map(dic_maker,lista_link))
```

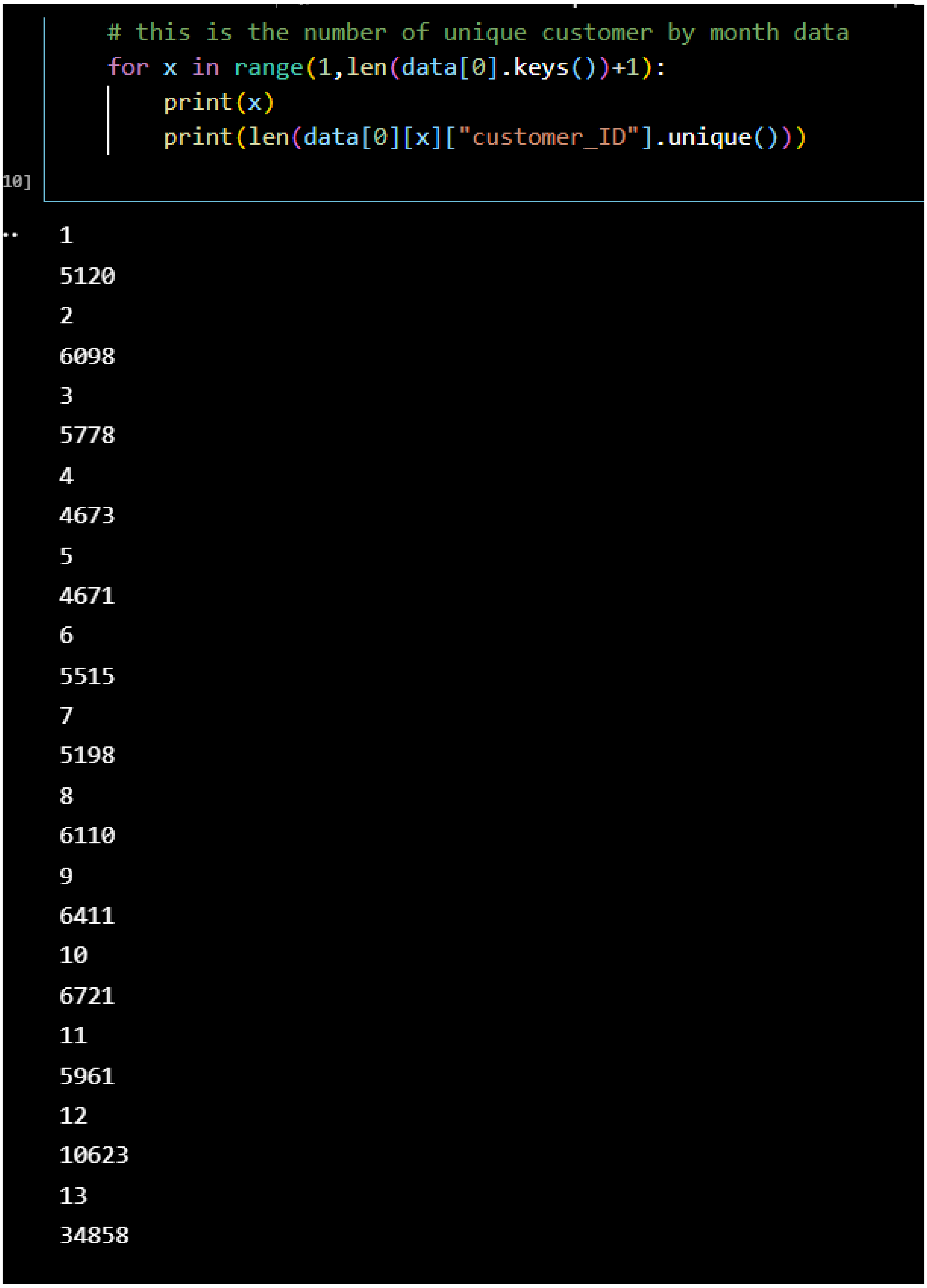
this is a funcion to delet entry in the dicionary as they are being group by month

```
def del_dic(data,num_key,num):

    del data[num][num_key]
    if num==len(data)-1:
        return data
    else:
        num+=1
        return del_dic(data,num_key,num)
```

this his how i concatonate all entries by month together with the exeption of the 13 month data

```
def concat_dic(data,arr_key):

    num_key=arr_key[0]

    data[0][num_key]=pd.concat(list(map(lambda x : x[num_key] , data)))

    num=1

    data=del_dic(data,num_key,num)

    arr_key.pop(0)

    if len(arr_key)==0:
        return data

    else:
        return concat_dic(data,arr_key)
```

after doing this I can check how many unique customers exist per month



```
# this is the number of unique customer by month data
for x in range(1,len(data[0].keys())+1):
    print(x)
    print(len(data[0][x]["customer_ID"].unique()))
```

```
10]
```

```
1
5120
2
6098
3
5778
4
4673
5
4671
6
5515
7
5198
8
6110
9
6411
10
6721
11
5961
12
10623
13
34858
```

here I take out all columns with more then 75% of NAN

```
def change_col(link):

    data_1=pd.read_csv(link)
    data_1=change_type(data_1)

    lista_col=list(filter(lambda x: x if
(data_1[x].isnull().sum()/len(data_1[x]))*100>=75 else None,data_1.columns))

    data_1.drop(labels=lista_col,axis=1,inplace=True)

    data_1.drop(labels=['Unnamed: 0.1', 'Unnamed: 0'],axis=1,inplace=True)

    data_1.to_csv(link)

    return
```

and use with threads

```
with cf.ThreadPoolExecutor() as excutor:
    list(excutor.map(change_col,link_list))
```

now to make this beta model I will reduce the dimentions of the data to the last month entrie from the one month data to the 7 month data, I will also grab the 13 month data not to use it in the model but for comparing the data, considering that the 13 month is the bulk of the data.

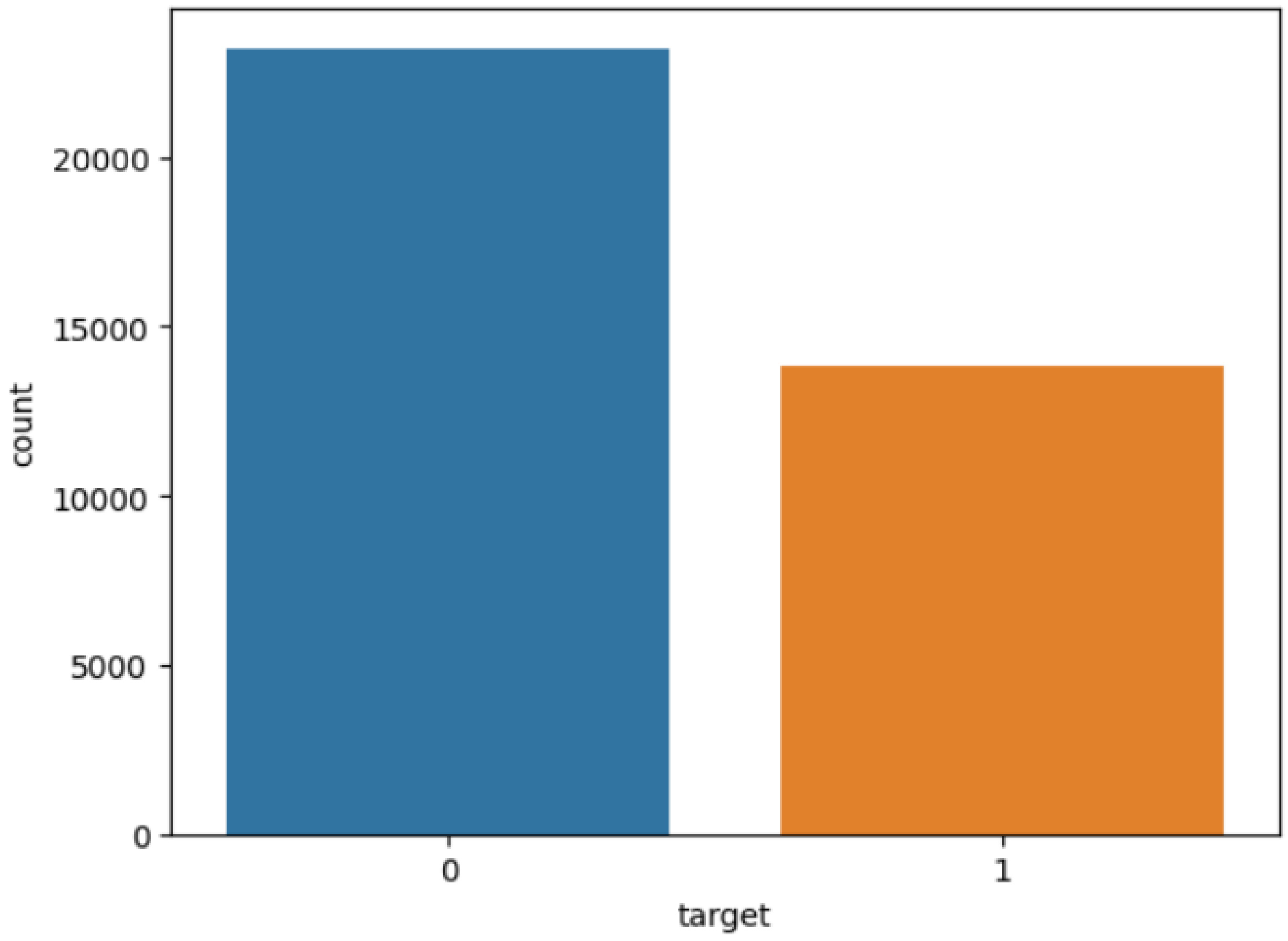
```
def get_last(X,ID):
    return X[X["customer_ID"]==ID].sort_values("S_2").iloc[-1,:]

X_1=1_month_data.iloc[:,1:]
X_2=2_month_data.iloc[:,1:]
X_3=3_month_data.iloc[:,1:]
X_4=4_month_data.iloc[:,1:]
X_5=5_month_data.iloc[:,1:]
X_6=6_month_data.iloc[:,1:]
X_7=7_month_data.iloc[:,1:]

X=pd.concat([X_2,X_3,X_4,X_5,X_6,X_7])


X_13=X_13.iloc[:,1:]




del X_2,X_3,X_4,X_5,X_6,X_7

with cf.ThreadPoolExecutor() as excutor:
    X=pd.concat(list(excutor.map(lambda
x:get_last(X,x),X["customer_ID"].unique())),axis=1)

X=X.T

X=pd.concat([X,X_1])
```

```
del X_1
```

View Data

Now that we finally have the data that is going to be the subject of study lets check the distribuition of the target

```
y=y.groupby("target")["target"].count()
y=y.to_frame().rename(columns={"target":"count"}).reset_index()
sns.barplot(x=y["target"],y=y["count"])
```



Has it can be seen the target is not balanced something that is need to better train the model, on that note I did a sampling using this funcion

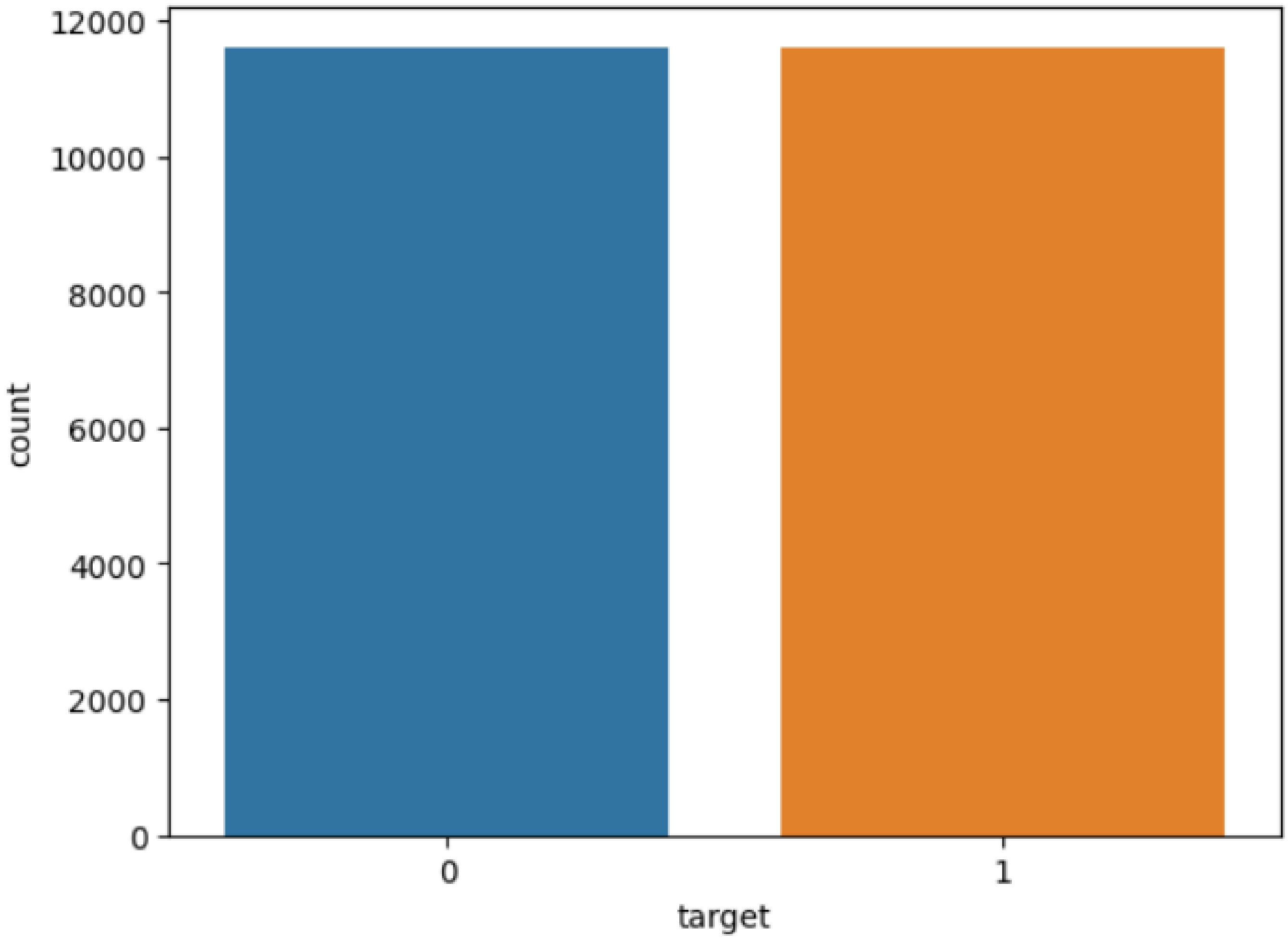
```
def equ_train(X_data,Y_data,equ,dis):

    data=pd.merge(X_data,Y_data,how="inner",on="customer_ID")
    X=data.iloc[:,:-1]
    y=data.iloc[:,-1]
    y=y.to_frame()
    customer_ID=data.iloc[:,0]


    y_1=y[y["target"]==dis]
    y_0=y[y["target"]==equ]

    y_1_half=y_1.iloc[:int(len(y_0)/2)]
    y_0_half=y_0.iloc[:int(len(y_0)/2)]
    y_train=pd.concat([y_0_half,y_1_half])
    y_1_half=y_1.iloc[int(len(y_0)/2):]
    y_0_half=y_0.iloc[int(len(y_0)/2):]
    y_test=pd.concat([y_0_half,y_1_half])
    X_train=X[X.index.isin(y_train.index)]
    X_test=X[X.index.isin(y_test.index)]

    return X_train,X_test,y_train,y_test
```

this funcion will return an X_train and y_train that will be balanced so I use it this way

```
equi_X,_,equi_y,_=ut.equ_train(X,y,0,1)
```

the "test" would be unbalanced, so no need to keep it for the model no lets check the distribution of the target data

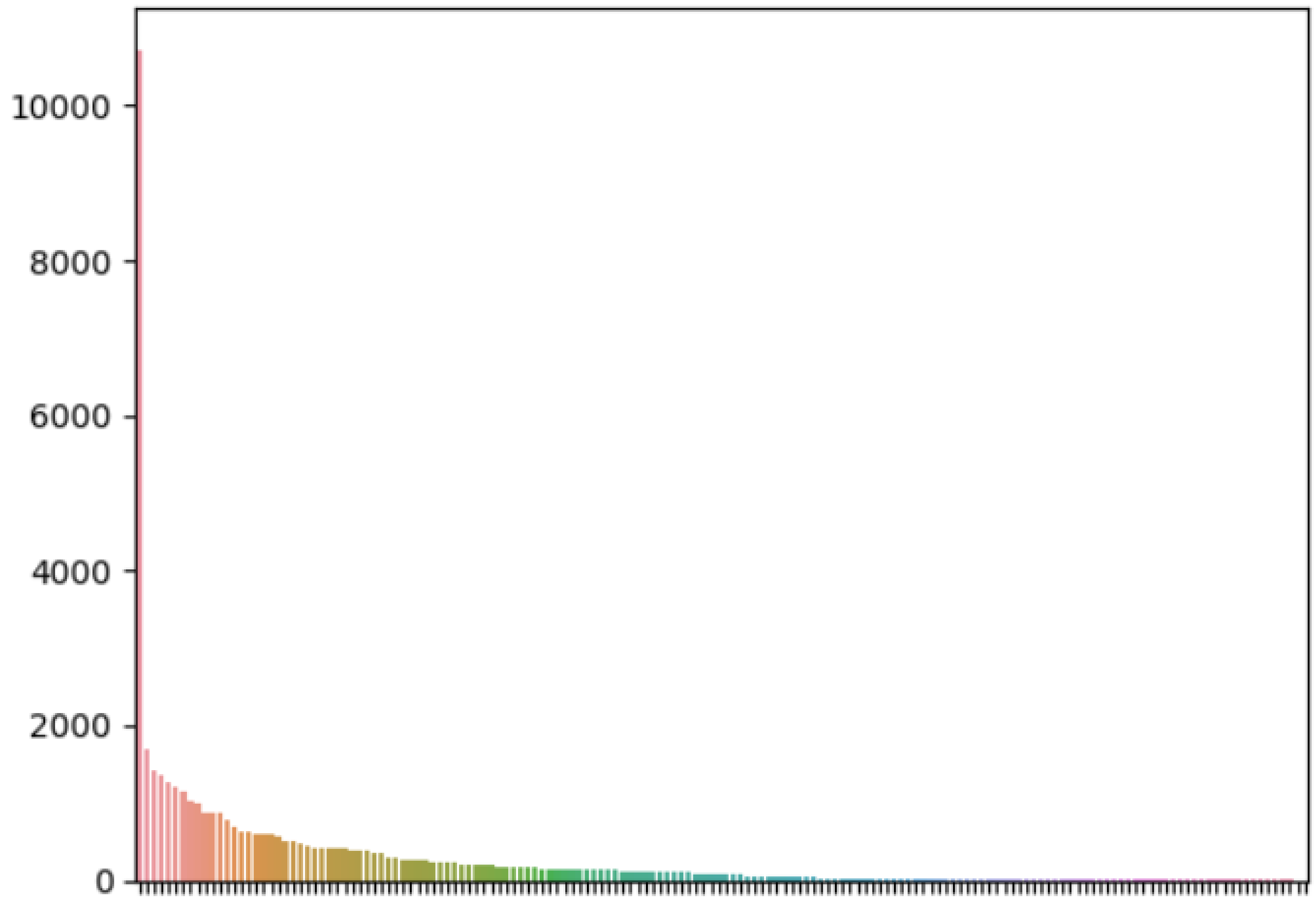
```
equi_y=equi_y.groupby("target")["target"].count()
equi_y=equi_y.to_frame().rename(columns={"target":"count"}).reset_index()
sns.barplot(x=equi_y["target"],y=equi_y["count"])
```



Due to the fact that the data has 190 dimention, aka columns, it is needed to reduce them for a better understanding of the data and also to avoid overfitting. To do this it is important first to make a train test split to avoid contaminating the test data, then input data where nan's are present and finally using a SelectKBest() to determind the best columns to use, mostly following the rule of log(number_of_observations)*5=number of columns. But first lets see the relative importance of all columns graficaly

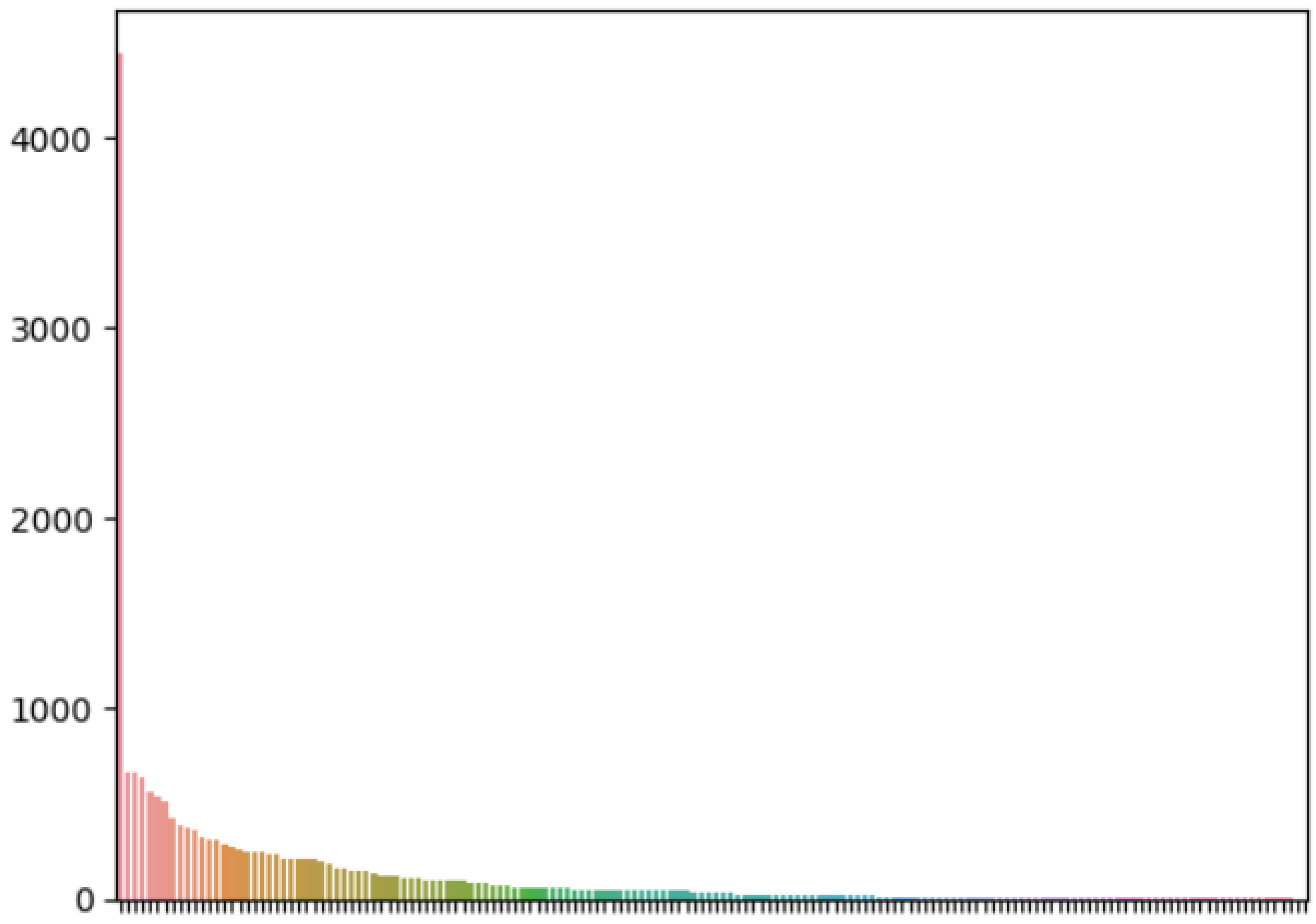
```
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=.3,random_state=42)

categorical=list(X_train.select_dtypes("object").columns)
numerical=list(X_train.select_dtypes("number").columns)

cat_pipe=Pipeline([
    ("imputer",SimpleImputer(strategy="most_frequent", missing_values=np.nan)),

("encoder",OrdinalEncoder(handle_unknown="use_encoded_value",unknown_value=-1)),
    ("scaler",StandardScaler())
    ])

num_pipe=Pipeline([
("imputer",KNNImputer(n_neighbors=5,weights="uniform",missing_values=np.nan)),
("scaler",StandardScaler())
])

preprocess=ColumnTransformer([("cat",cat_pipe,categorical),
("num",num_pipe,numerical)])
```

```
preprocess.fit(X_train)

X_train=preprocess.transform(X_train)

skb=SelectKBest(k="all")
skb.fit(X_train,y_train.values.reshape(1,-1)[0])

imp_feat=pd.DataFrame(skb.scores_).sort_values(0,ascending=False)
graph=sns.barplot(x=imp_feat.index,y=imp_feat[0], order=imp_feat.index)
graph.set(xticklabels=[]);
```



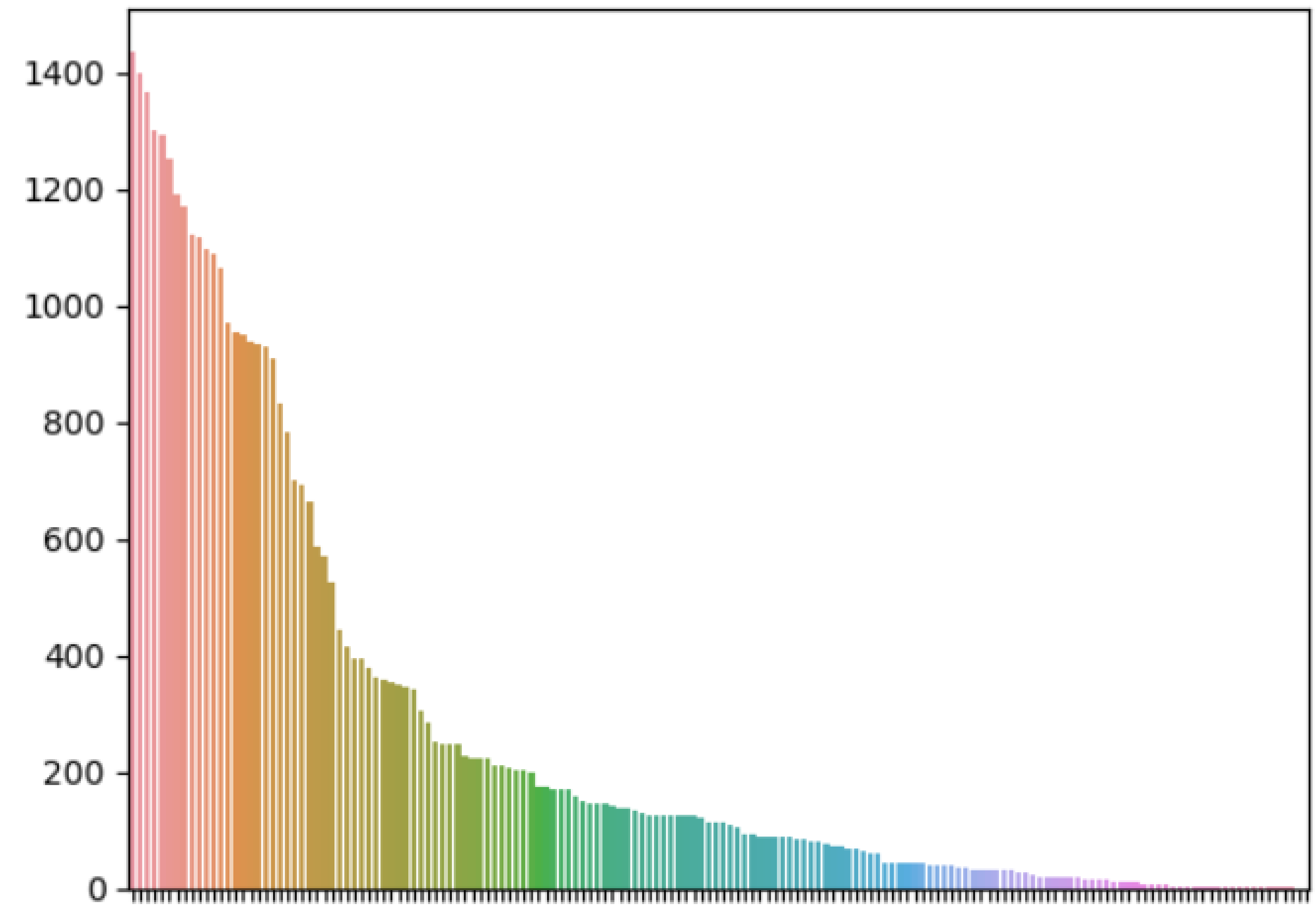
Has it can be seen this data has one single column that has an extreme value of importance relitive to others, this type of behaviour in the data can be a blessing or a curse, as the data we are analizing could have a caracteristic that will not be found in other data. To insure that I did the same quick view in the test data

```
skb=SelectKBest(k="all")
skb.fit(preprocess.transform(X_test),y_test.values.reshape(1,-1)[0])
imp_feat=pd.DataFrame(skb.scores_).sort_values(0,ascending=False)
graph=sns.barplot(x=imp_feat.index,y=imp_feat[0], order=imp_feat.index)
graph.set(xticklabels=[]);
```



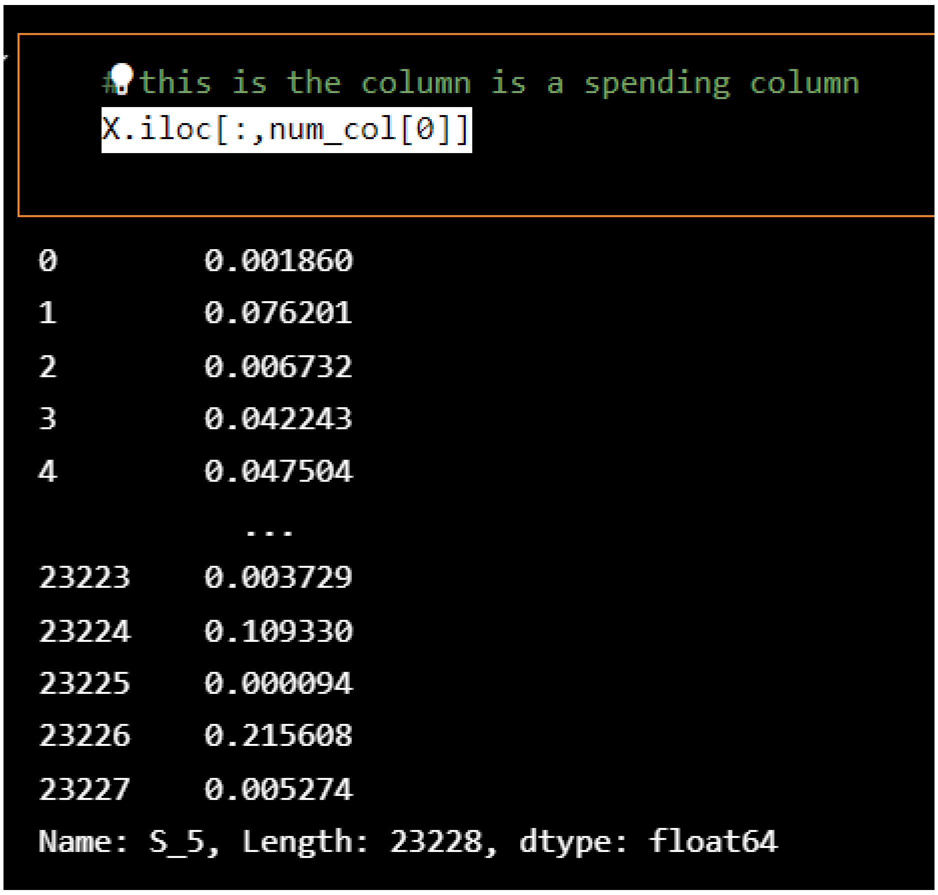
here we see that it also happens although not to the same extend, so just to be sure I will check the 13 months data even though I will not be in this project working with it

```
temp=preprocess.transform(X_13)
skb=SelectKBest(k="all")
skb.fit(temp,y_great.values.reshape(1,-1)[0])
imp_feat=pd.DataFrame(skb.scores_).sort_values(0,ascending=False)
graph=sns.barplot(x=imp_feat.index,y=imp_feat[0], order=imp_feat.index)
graph.set(xticklabels=[]);
```



my suspicions were confirm in the 13 month data, as it can be seen there is no colum that beaves in the same way and because of that I will resign that column to not be use for predictive measurement, to do this I have to id the column

```
skb=SelectKBest(k=1)
skb.fit(X_train,y_train.values.reshape(1,-1)[0])
data_X=skb.transform(X_train)

num_col=list(filter(lambda x: x if
data_X.tolist()==X_train[:,x].reshape(-1,1).tolist() else None,range(0,159)))

X.iloc[:,num_col[0]]
```

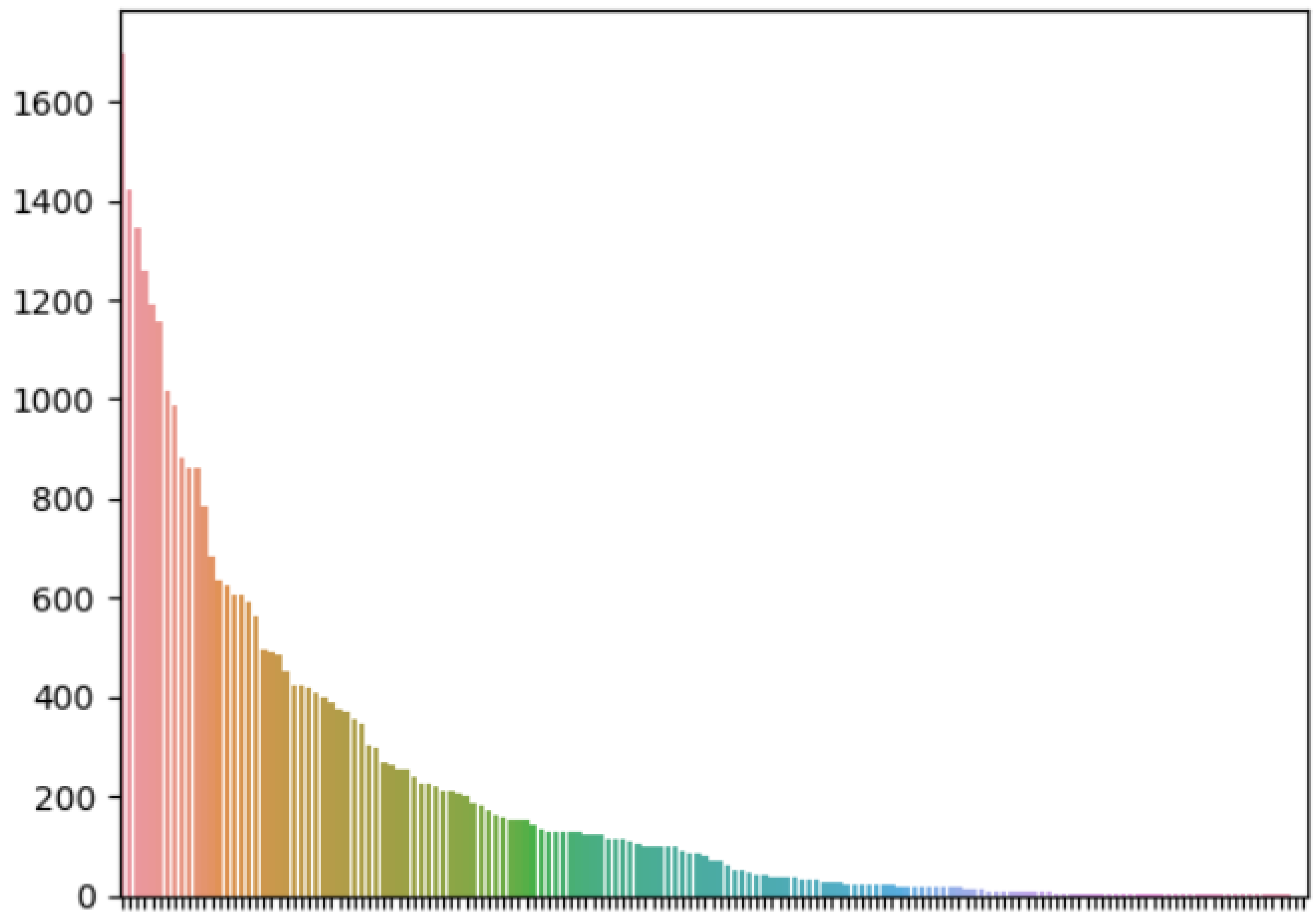


```
#this is the column is a spending column
X.iloc[:,num_col[0]]
```

```
0        0.001860
1        0.076201
2        0.006732
3        0.042243
4        0.047504
           ...
23223    0.003729
23224    0.109330
23225    0.000094
23226    0.215608
23227    0.005274
Name: S_5, Length: 23228, dtype: float64
```

Has we can see, the column was a spending column, now lests take it out and see

```
X_train=pd.DataFrame(X_train)
X_train=X_train.drop(num_col[0],axis=1)
skb=SelectKBest(k="all")
skb.fit(X_train,y_train.values.reshape(1,-1)[0])
imp_feat=pd.DataFrame(skb.scores_).sort_values(0,ascending=False)
graph=sns.barplot(x=imp_feat.index,y=imp_feat[0], order=imp_feat.index)
graph.set(xticklabels=[]);
```



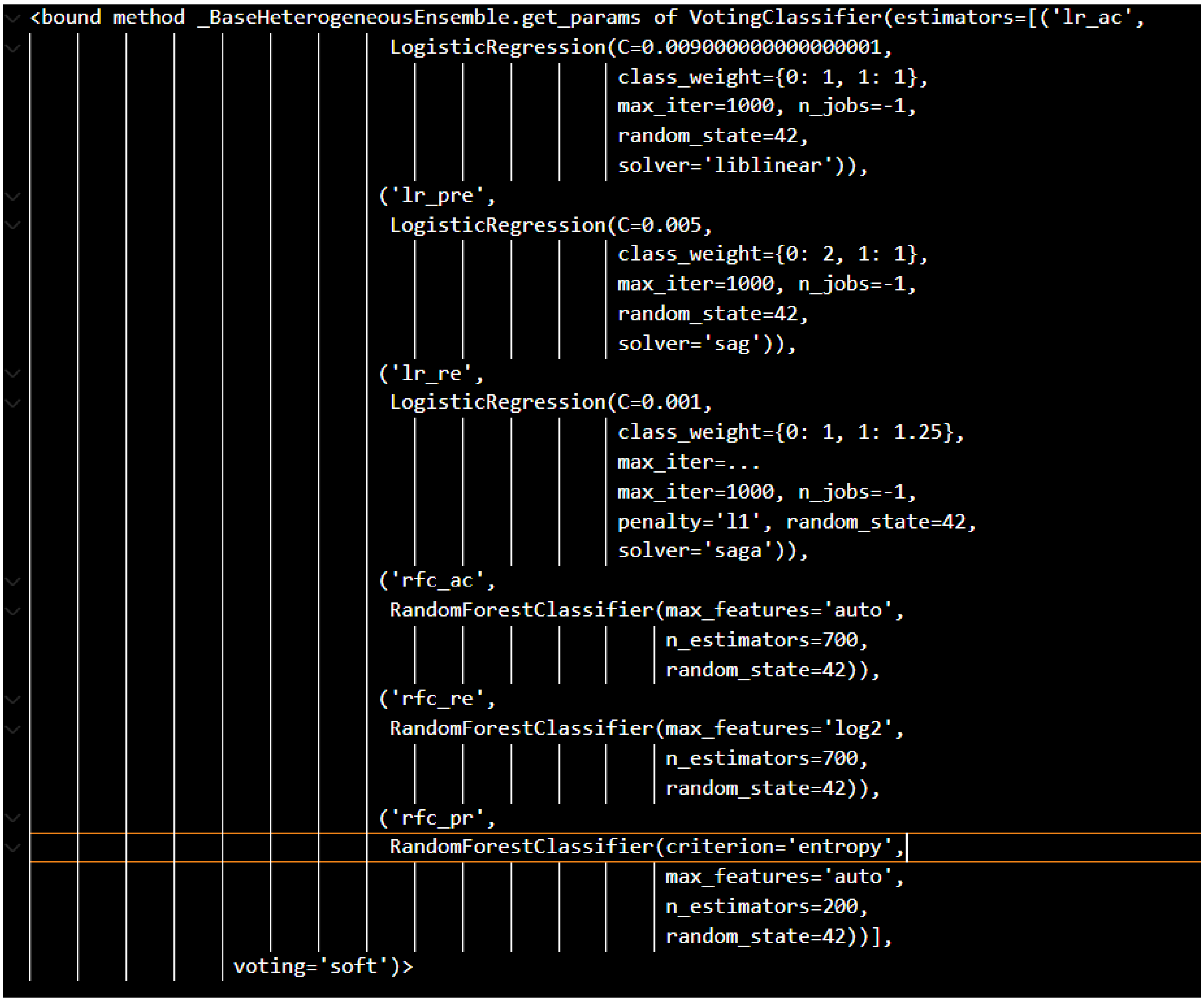
Now it resembles much more the the 13 month column, to top it of lets get the top 25 column that remain and both in test and train

```
skb=SelectKBest(k=25)
skb.fit(X_train,y_train)
X_train=skb.transform(X_train)
X_test=skb.transform(X_test)
X_great=skb.transform(X_great)
```

make a pipe

```
final_pre_pipe=Pipeline([("preprocess",preprocess),("skb",skb)])
```

once the preprocess pipe was made the models were train using diferent metrics for perfomance accuracy, precision and recall esulting in this models



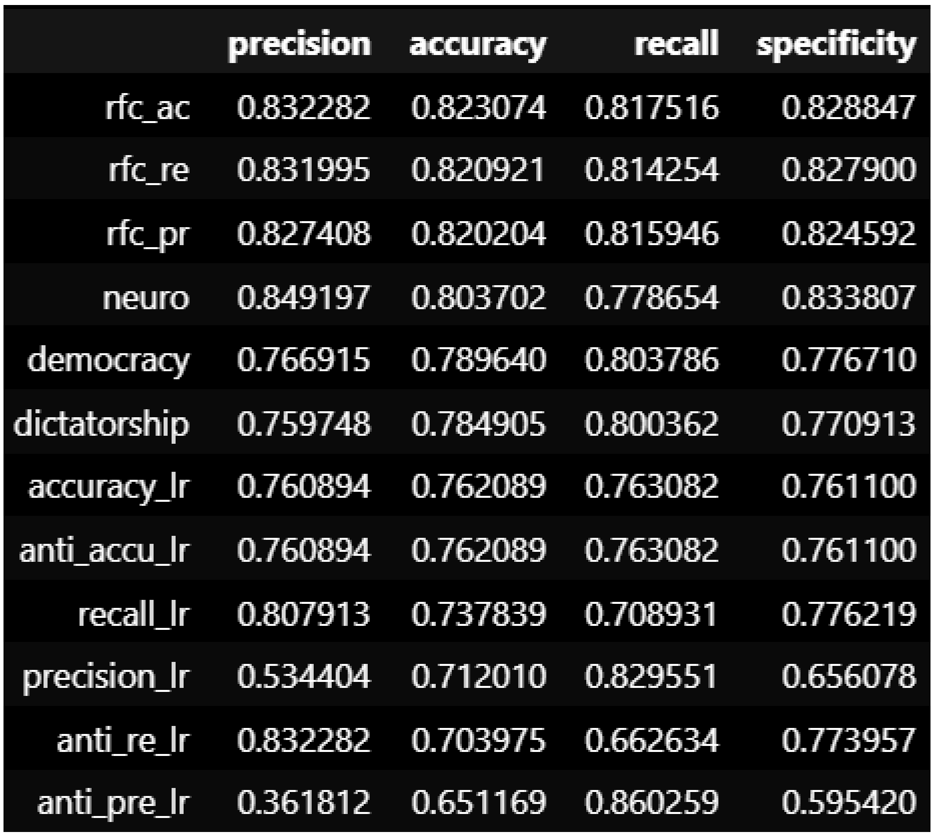
```
<bound method _BaseHeterogeneousEnsemble.get_params of VotingClassifier(estimators=[('lr_ac',
                              LogisticRegression(C=0.009000000000000001,
                                                 class_weight={0: 1, 1: 1},
                                                 max_iter=1000, n_jobs=-1,
                                                 random_state=42,
                                                 solver='liblinear')),
                             ('lr_pre',
                              LogisticRegression(C=0.005,
                                                 class_weight={0: 2, 1: 1},
                                                 max_iter=1000, n_jobs=-1,
                                                 random_state=42,
                                                 solver='sag')),
                             ('lr_re',
                              LogisticRegression(C=0.001,
                                                 class_weight={0: 1, 1: 1.25},
                                                 max_iter=...
                                                 max_iter=1000, n_jobs=-1,
                                                 penalty='l1', random_state=42,
                                                 solver='saga')),
                             ('rfc_ac',
                              RandomForestClassifier(max_features='auto',
                                                     n_estimators=700,
                                                     random_state=42)),
                             ('rfc_re',
                              RandomForestClassifier(max_features='log2',
                                                     n_estimators=700,
                                                     random_state=42)),
                             ('rfc_pr',
                              RandomForestClassifier(criterion='entropy',
                                                     max_features='auto',
                                                     n_estimators=200,
                                                     random_state=42))],
                 voting='soft')>
```

theses models were then also place in a ensemble voting system both hard and soft. and neuro-network model was also developt

```
model=keras.Sequential([
    layers.Dense(50,activation="relu",input_shape=[27]),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),
    layers.Dense(40,activation="relu"),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),
    layers.Dense(30,activation="relu"),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),
    layers.Dense(20,activation="relu"),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),
    layers.Dense(10,activation="relu"),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),
    layers.Dense(1,activation="sigmoid"),
])
```

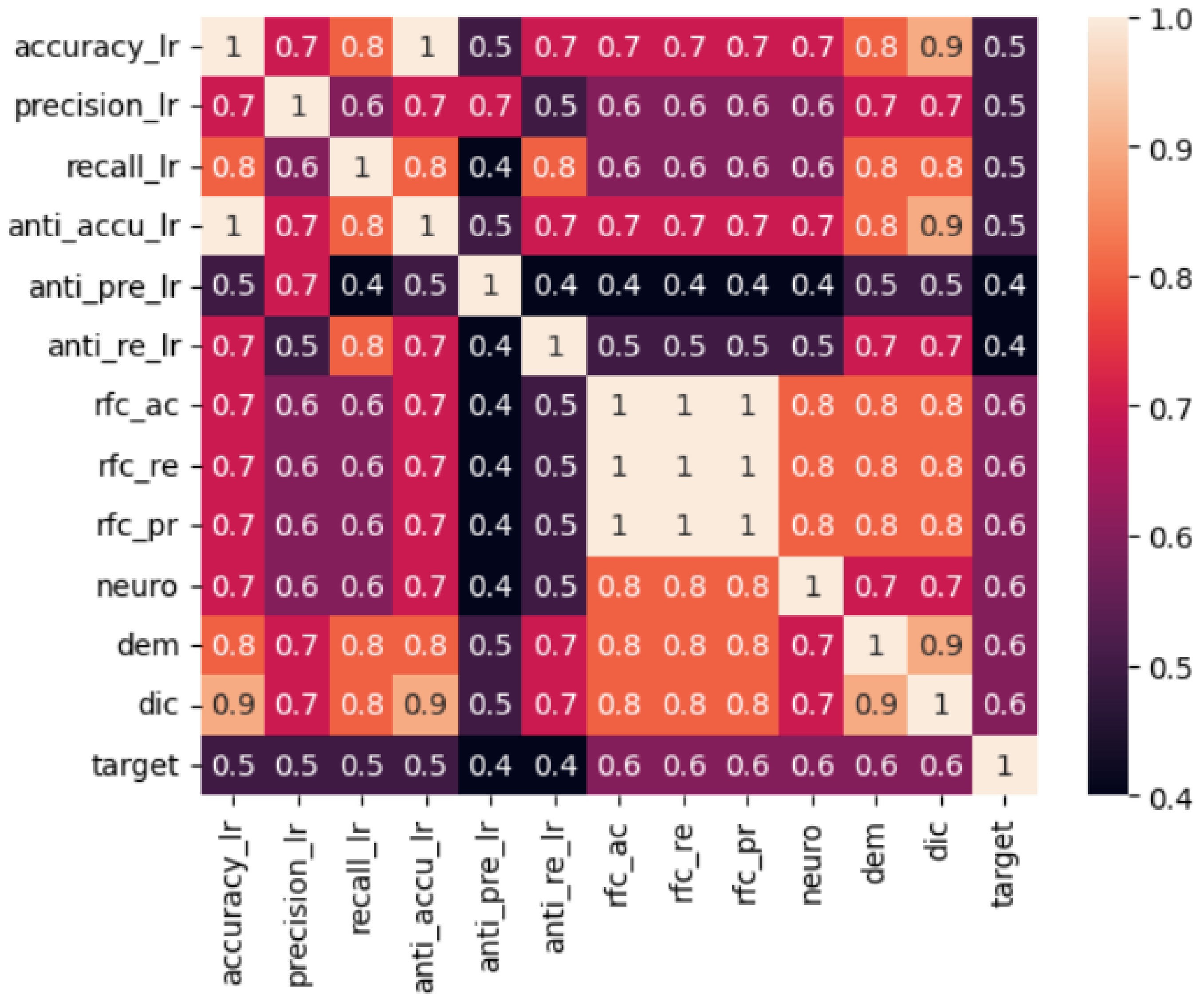
```
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["binary_accuracy"]
)

early_stopping=keras.callbacks.EarlyStopping(
    patience=10,
    min_delta=0.001,
    restore_best_weights=True
)

history1= model.fit(
    X_train_new,y_train_new,
    validation_data=(X_valid_new,y_valid_new),
    batch_size=2000,
    epochs=1000,
    callbacks=[early_stopping],
    verbose=0
)
```

after the models where fited to the training data this was the result in test

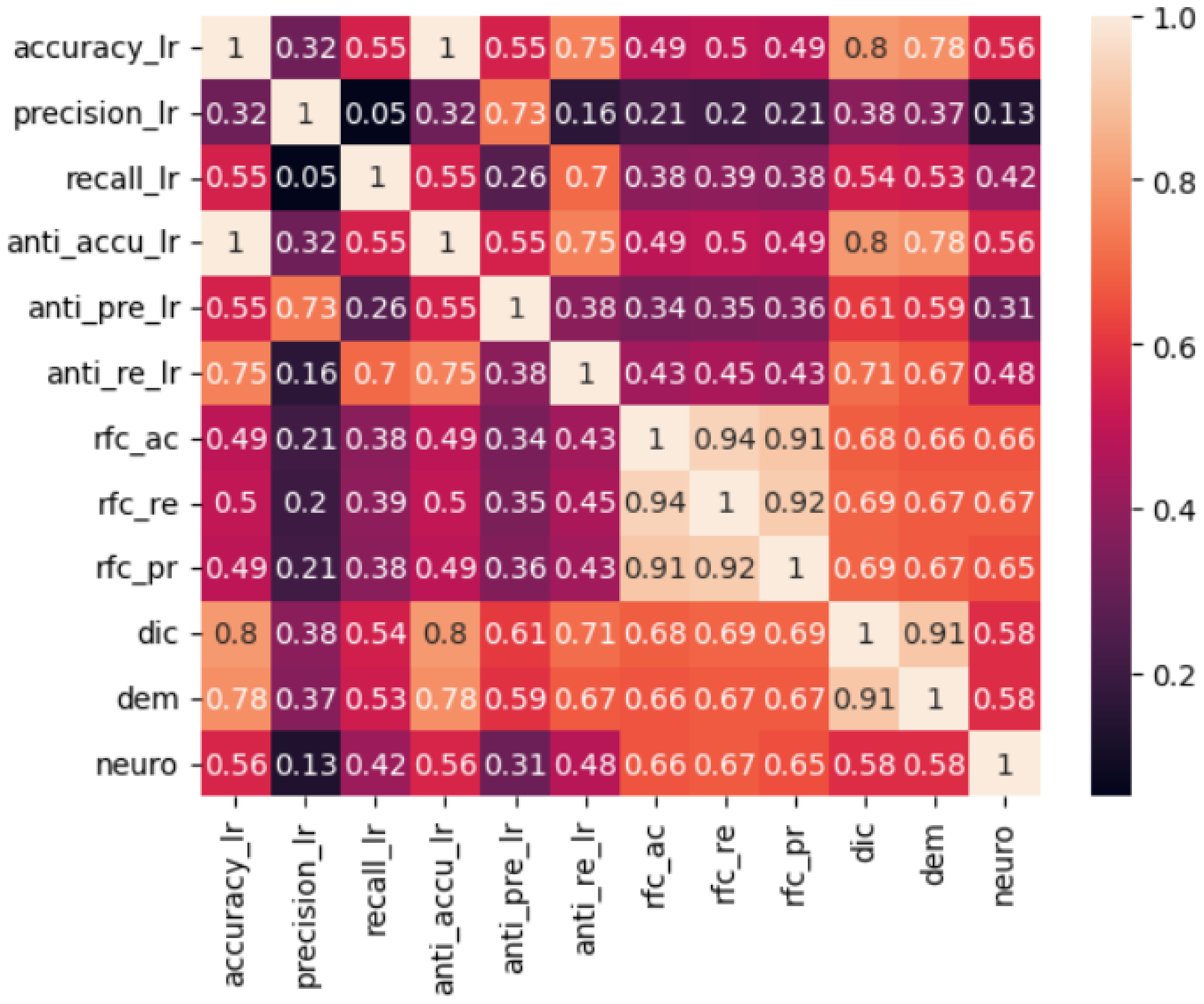


| | precision | accuracy | recall | specificity |
|---|---|---|---|---|
| rfc_ac | 0.832282 | 0.823074 | 0.817516 | 0.828847 |
| rfc_re | 0.831995 | 0.820921 | 0.814254 | 0.827900 |
| rfc_pr | 0.827408 | 0.820204 | 0.815946 | 0.824592 |
| neuro | 0.849197 | 0.803702 | 0.778654 | 0.833807 |
| democracy | 0.766915 | 0.789640 | 0.803786 | 0.776710 |
| dictatorship | 0.759748 | 0.784905 | 0.800362 | 0.770913 |
| accuracy_lr | 0.760894 | 0.762089 | 0.763082 | 0.761100 |
| anti_accu_lr | 0.760894 | 0.762089 | 0.763082 | 0.761100 |
| recall_lr | 0.807913 | 0.737839 | 0.708931 | 0.776219 |
| precision_lr | 0.534404 | 0.712010 | 0.829551 | 0.656078 |
| anti_re_lr | 0.832282 | 0.703975 | 0.662634 | 0.773957 |
| anti_pre_lr | 0.361812 | 0.651169 | 0.860259 | 0.595420 |

Here we have the correlation of the perdictions to the target and to each other



And here we have the correlations of the models correct model predictions to each other



| | accuracy_lr | precision_lr | recall_lr | anti_accu_lr | anti_pre_lr | anti_re_lr | rfc_ac | rfc_re | rfc_pr | dic | dem | neuro |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| accuracy_lr | 1 | 0.32 | 0.55 | 1 | 0.55 | 0.75 | 0.49 | 0.5 | 0.49 | 0.8 | 0.78 | 0.56 |
| precision_lr | 0.32 | 1 | 0.05 | 0.32 | 0.73 | 0.16 | 0.21 | 0.2 | 0.21 | 0.38 | 0.37 | 0.13 |
| recall_lr | 0.55 | 0.05 | 1 | 0.55 | 0.26 | 0.7 | 0.38 | 0.39 | 0.38 | 0.54 | 0.53 | 0.42 |
| anti_accu_lr | 1 | 0.32 | 0.55 | 1 | 0.55 | 0.75 | 0.49 | 0.5 | 0.49 | 0.8 | 0.78 | 0.56 |
| anti_pre_lr | 0.55 | 0.73 | 0.26 | 0.55 | 1 | 0.38 | 0.34 | 0.35 | 0.36 | 0.61 | 0.59 | 0.31 |
| anti_re_lr | 0.75 | 0.16 | 0.7 | 0.75 | 0.38 | 1 | 0.43 | 0.45 | 0.43 | 0.71 | 0.67 | 0.48 |
| rfc_ac | 0.49 | 0.21 | 0.38 | 0.49 | 0.34 | 0.43 | 1 | 0.94 | 0.91 | 0.68 | 0.66 | 0.66 |
| rfc_re | 0.5 | 0.2 | 0.39 | 0.5 | 0.35 | 0.45 | 0.94 | 1 | 0.92 | 0.69 | 0.67 | 0.67 |
| rfc_pr | 0.49 | 0.21 | 0.38 | 0.49 | 0.36 | 0.43 | 0.91 | 0.92 | 1 | 0.69 | 0.67 | 0.65 |
| dic | 0.8 | 0.38 | 0.54 | 0.8 | 0.61 | 0.71 | 0.68 | 0.69 | 0.69 | 1 | 0.91 | 0.58 |
| dem | 0.78 | 0.37 | 0.53 | 0.78 | 0.59 | 0.67 | 0.66 | 0.67 | 0.67 | 0.91 | 1 | 0.58 |
| neuro | 0.56 | 0.13 | 0.42 | 0.56 | 0.31 | 0.48 | 0.66 | 0.67 | 0.65 | 0.58 | 0.58 | 1 |

## Conclusion

Considering that the aim of the model is to prevent defaults on loan the best measurement to use is recall.
When looking at the first table with all the scoring metrics we see that the model with the highest recall is the logistic regression model that was train with the inverted target and a scoring metric for precision (anti_pre_lr), with a recall score in the testing data of 86%.
However we can see in the first correlation table that the correlation to the target is 0.4, still above a coin as that would be a value of 0 but worse then any of the other models with the exception of the inverse target model scored with recall (anit_re_lr).
Looking a bit further at the model's predictions we can see it's confusion matrix table were:

```
[[3276  205]
 [2226 1262]]
```

We can see that it's predictions have a bias for 1 as expected with a high recall value. This issue is also represented in the
fact that this model as a 65% accuracy in a balance binary test.
For that reason I would be more incline to use the Random Forest Classifier train at accuracy with the second highest recall of 81.7%.