FEATURE ENGINEERING

MONTH
The feature month has the highest importance in the XGBoost model, however in the exploratory data analysis we did not notice any trend.
Hence, we try to group the months based on the season.

The data show a trend:
In summer, the probability of yes drops, while it is at the highest in winter and spring, which are the two periods when people get extra money for various reasons.

AGE
the distribution drops at 60, according to the boxplot the fence it's at 70.
In order to obtain bins with comparable frequencies, we group as follow:
<30, >55 and in between we use bins of 5.

The data show a trend:
the probability of yes is high at the extrema, and it reduces in the middle of the distribution.
The right extrema is brought up by the outliers -> worth to explore that age range more

Note:
When encoding, use **label encoder** as the order of the categories matter.

DAY
We group the days as:
- payday: the most common days in which people receive salaries
- after payday: the three days after the reception of the salary
- the other days

With this grouping, the data show a clear trend:
The probability of Yes is highest on paydays, it decreases the days after and it drops in the other days

CAMPAIN
the distribution is very narrow as well -> I would group together the values >6 which is the upper fence in the box plot

The data show a clear trend: the probability of Yes reduces with the increase of the campaign number

DURATION
The duration distribution is wider than the balance, but with a long tail as well.
Given the low number of samples, I would group together all the durations above 900 (which is 15 minutes).
Then, I would create buckets of 60/120 seconds prior to that limit

The data show a clear trend: the longer the call, the higher the probability of yes.
This trend also makes sense logically

BALANCE
The distribution is very narrow: group together the samples <-2000 and >5000
in the middle of the distribution, create buckets of 500 or 1000
With the above described grouping, the data show a trend: the higher the balance, the

hihgher the chance of Yes. However, the top tier group has a lower Y probability, due to the outliers, but also the fact that wealthy people would invest in different financial tools.

JOB

We group together the following professions:

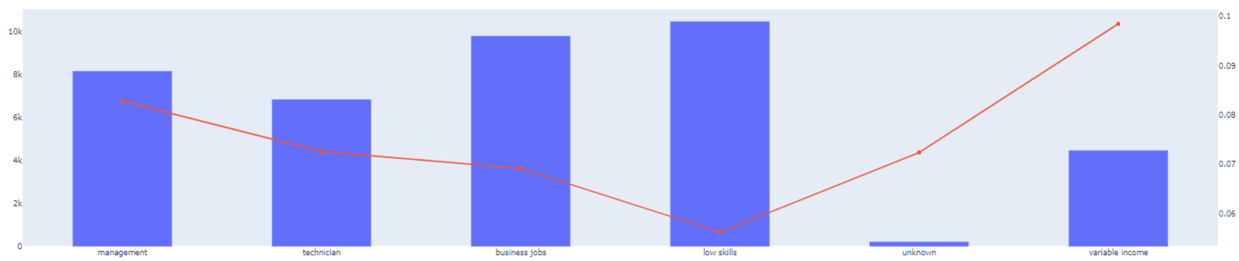
- Housemaid + blue collar → low skills
- Admin + services + entrepreneur → business jobs
- retired + self-employed + unemployed + student → no fixed income

The data show a trend: the higher the expected salary of the job, and the higher is the probability of Yes.

The category variable income, however, has the highest probability:

People with variable income tend to subscribe the deposit in order to earn some money.

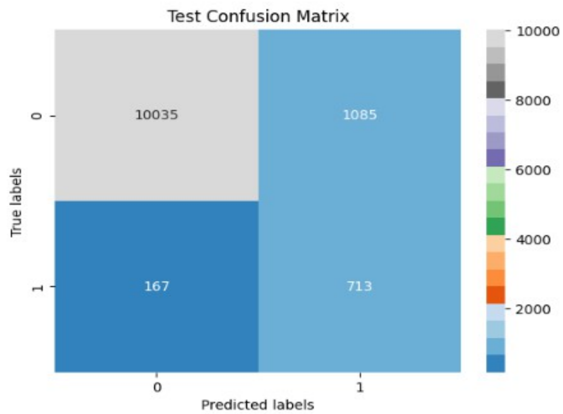
Frequency plot for attribute job and probability of Yes for each category



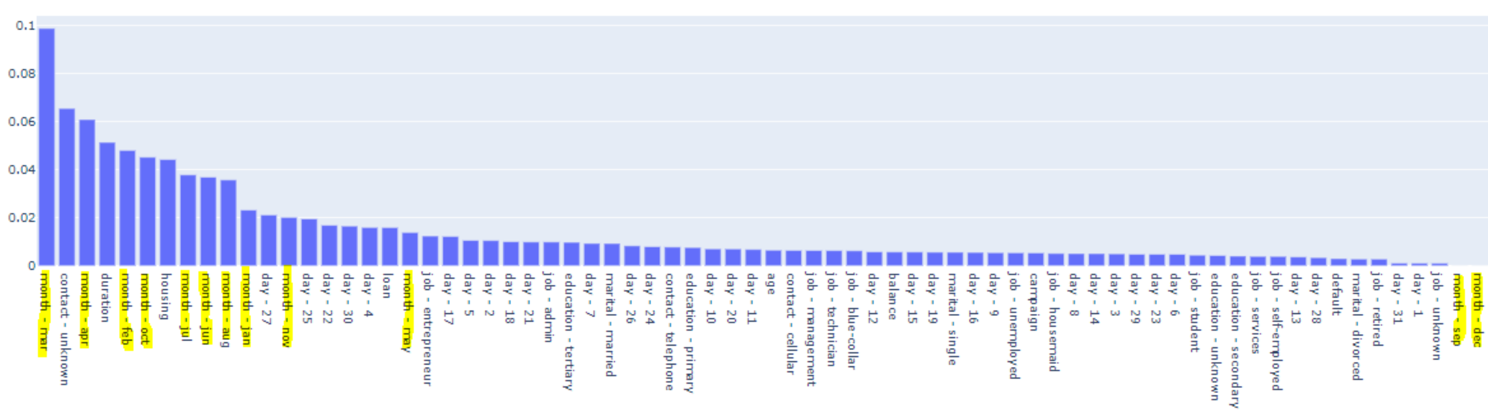
####### MODEL & PREFORMANCE #####

INITIAL TUNING

min_child_weight=10,
max_depth=4,
n_estimators=200,
scale_pos_weight=spw



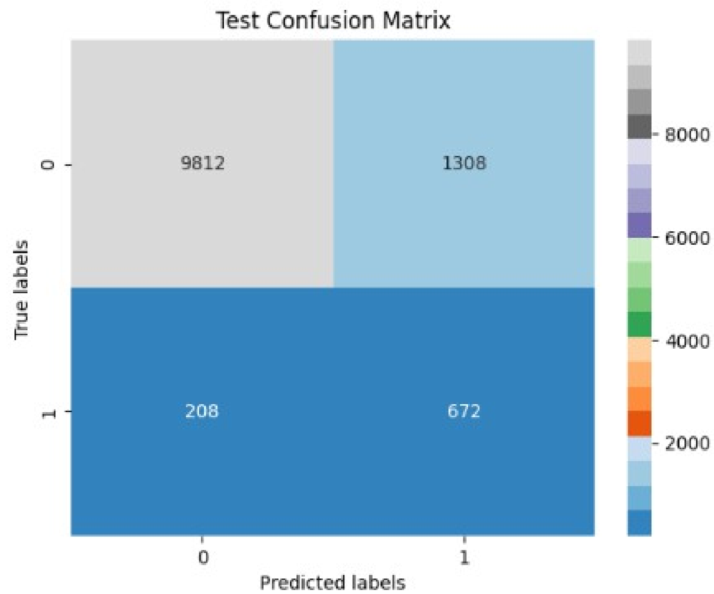

```
Train accuracy score: 0.9191428571428572
Test accuracy score: 0.8956666666666667
Train recall score: 0.9856150793650794
Test recall score: 0.8102272727272727
Train AUC score: 0.9498003044608648
Test AUC score: 0.8563276651406146
```



The feature month has high importance -> the model relies on it to take decisions.
However, in the data analysis we did not notice a strong correlation between the month and the label y

DROP FEATURE MONTH

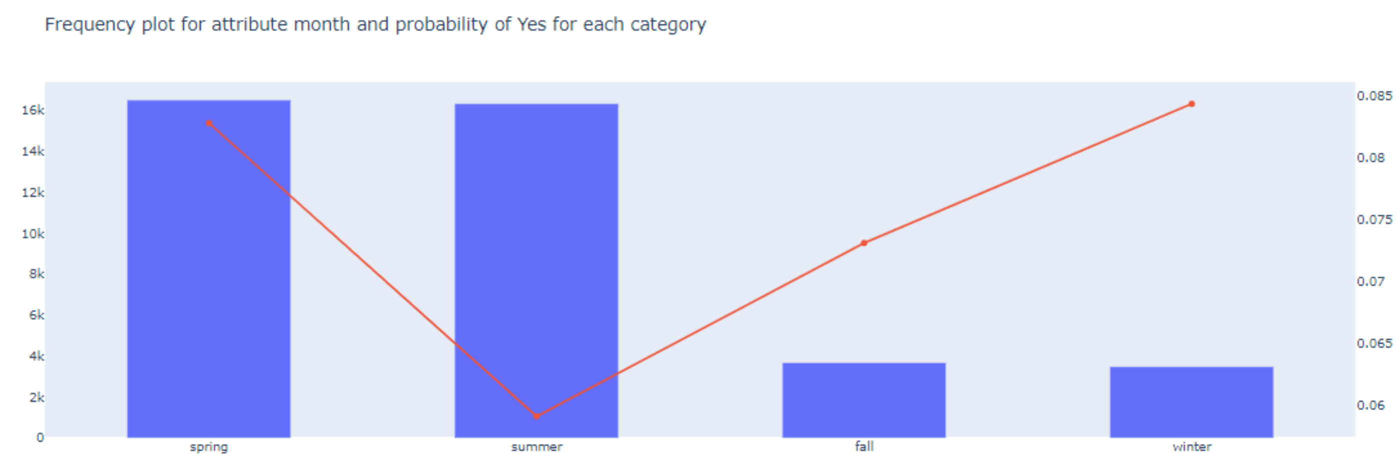
Let's try and drop the feature month and see how the model performs



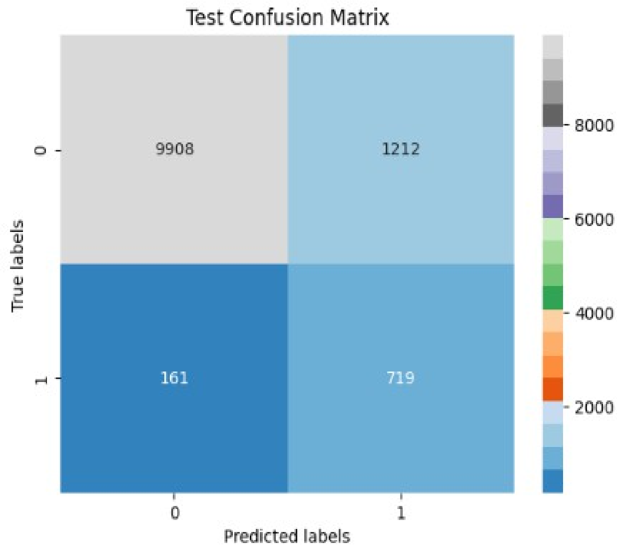

```
Train accuracy score: 0.8987857142857143
Test accuracy score: 0.8736666666666667
Train recall score: 0.966765873015873
Test recall score: 0.7636363636363637
Train AUC score: 0.9301386323207443
Test AUC score: 0.823005232177894
```

There is a **performance reduction**

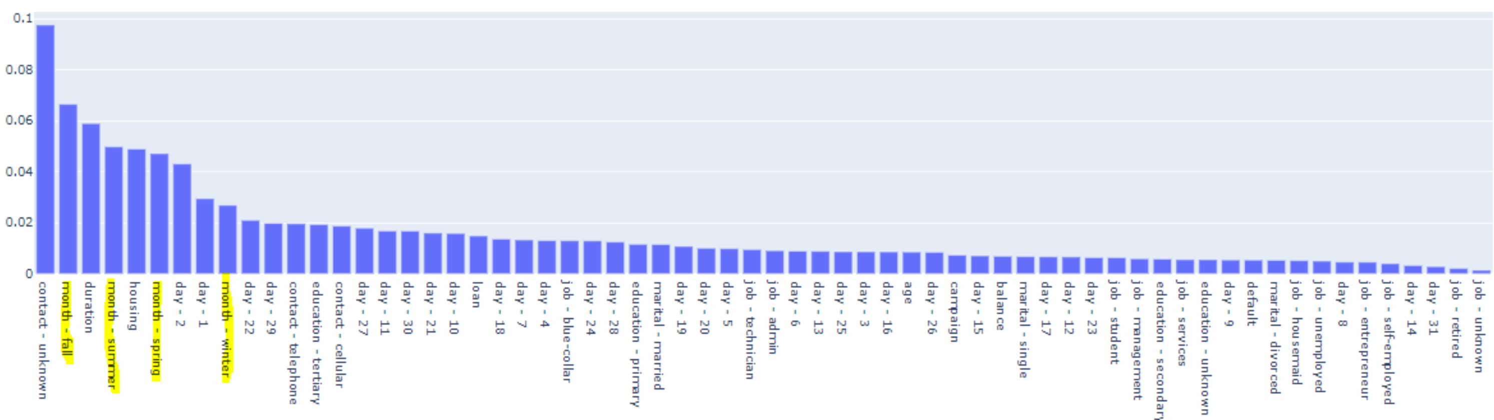
GROUP FEATURE MONTH
However, if we group the months based on the **season**:



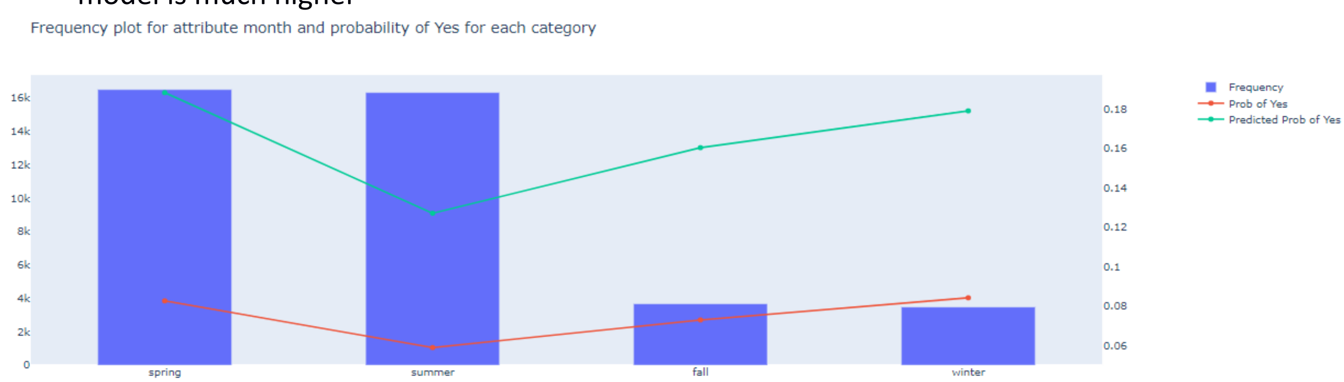
We notice that a correlation with Y is present:
- During summer the probability of Y is the lowest, because people spend money for vacations
- During spring and winter it's the highest (more or less at the same probability) because of the 13th month salary (winter) and the holiday allowance (14th month salary in spring)




```
Train accuracy score: 0.91
Test accuracy score: 0.8855833333333333
Train recall score: 0.9831349206349206
Test recall score: 0.8170454545454545
Train AUC score: 0.9437303297755882
Test AUC score: 0.8540263243950293
```

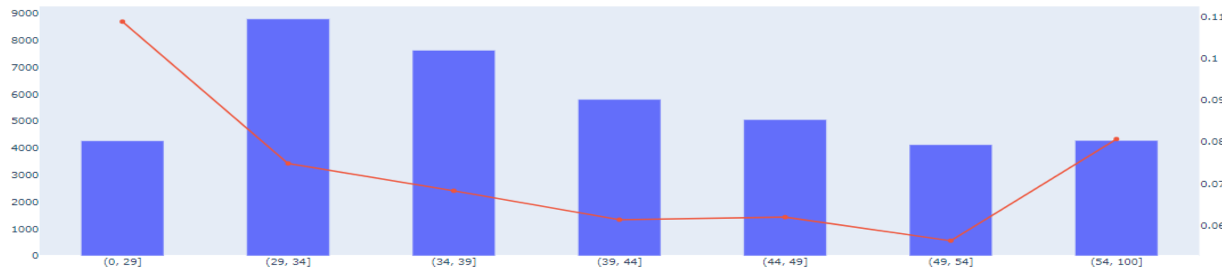


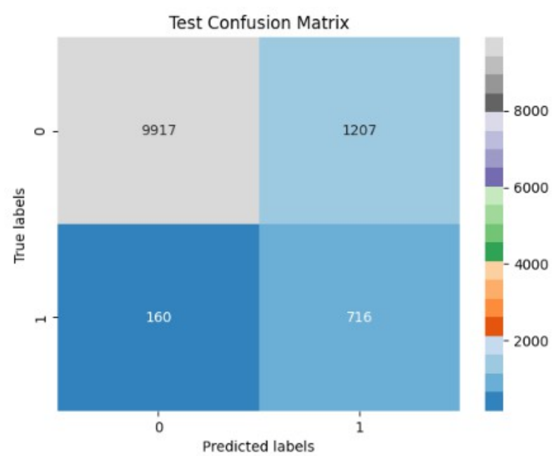
The performance are better than removing the feature -> there is actual information in the feautere. It is **slightly worse** than when the months were explicitly indicated, but now the interpretability of the model is much higher



We notice that the predicted values have the same trend as the real ones.
However, we can notice that overall the probability of Yes is higher in the prediction than in the true labels:
This is because we weight the Positive Weight and we are strongly interested in predicting the true Ys accepting some false positive.

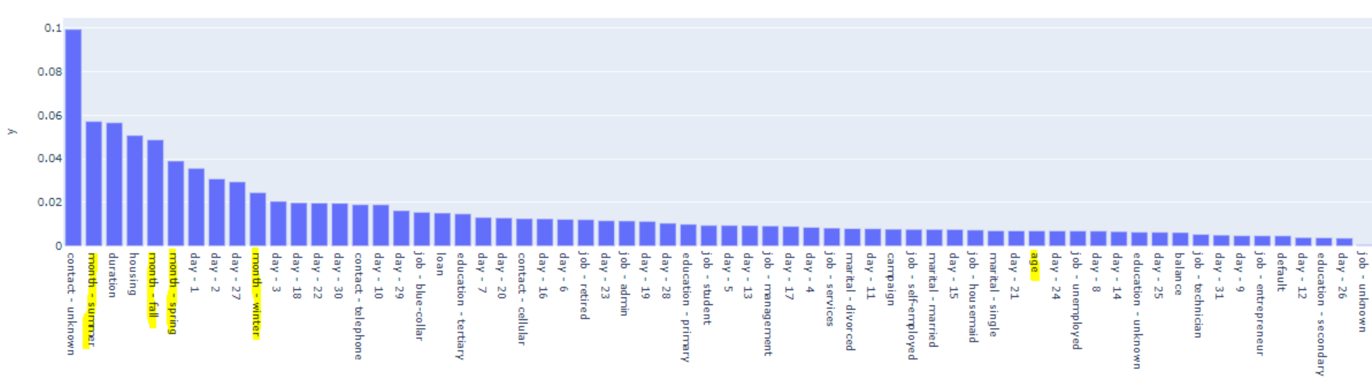
GROUP FEATURE AGE
We bin the age in ranges of 5 years, and we notice a pretty clear trend



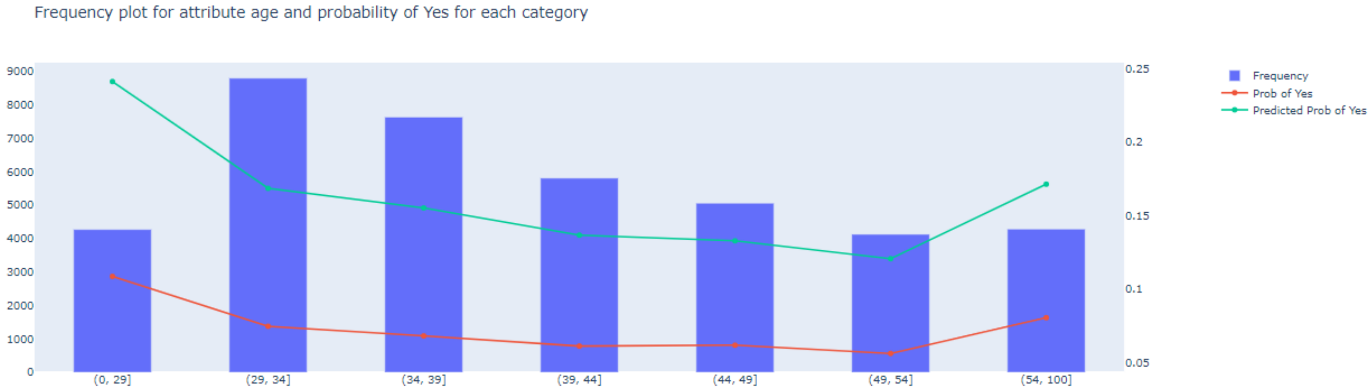



```
Train accuracy score: 0.91
Test accuracy score: 0.8855833333333333
Train recall score: 0.9831349206349206
Test recall score: 0.8170454545454545
Train AUC score: 0.9437303297755882
Test AUC score: 0.8540263243950293
```

The performance are comparable, the feature importance did not change much.
However, now the data are more interpretable.

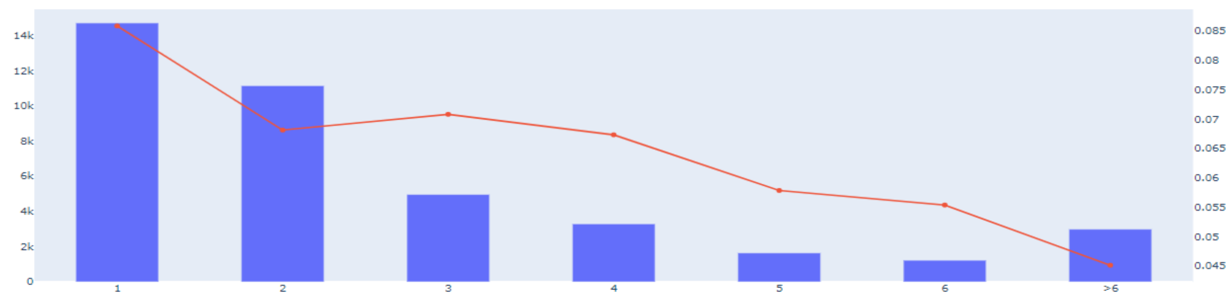


Once again the model picked up the trend, but the average Y predicted probability is higher because the positive samples are weighted

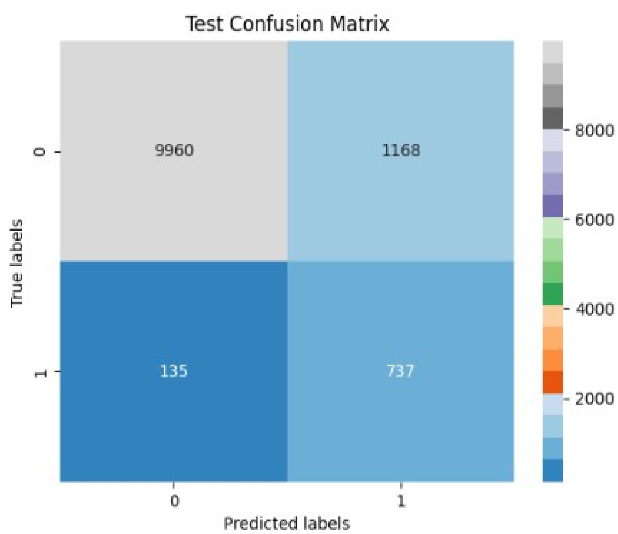


GROUP FEATURE CAMPAIGN

We cut off after 6, which is the upper fence in the box plot

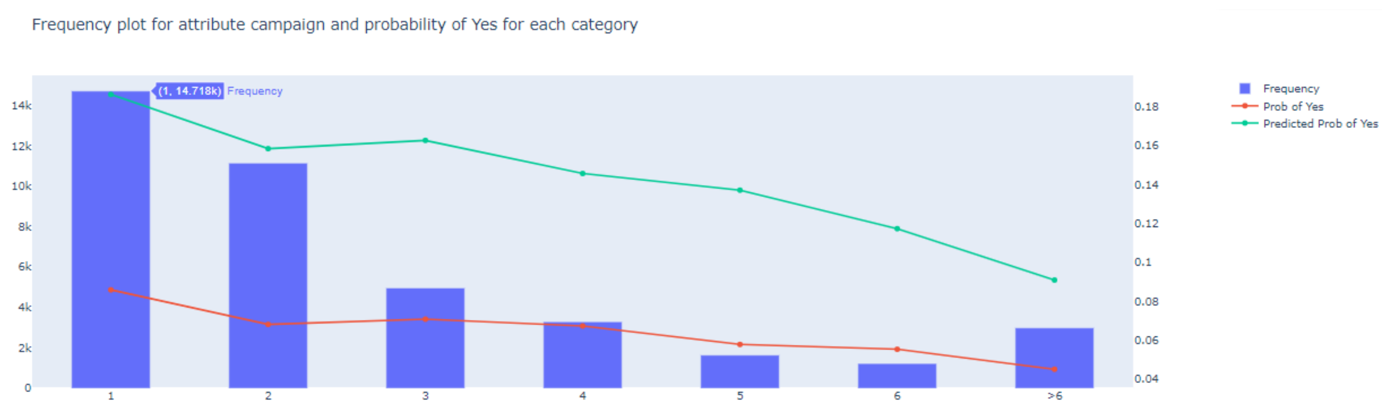


Now the data show a clear trend



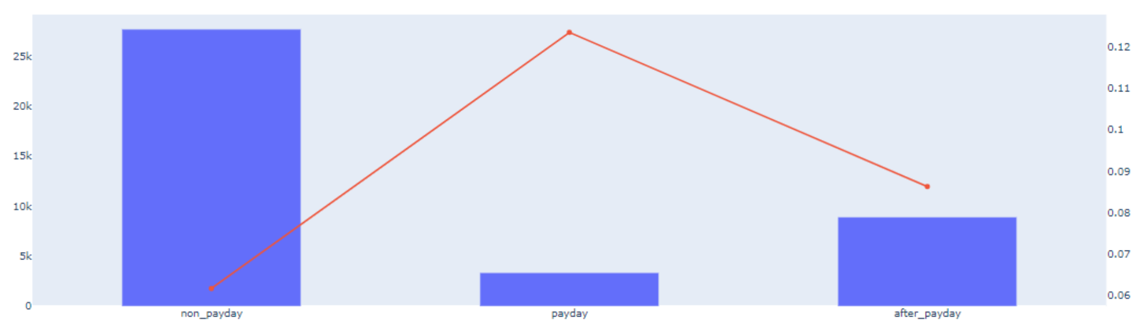

```
Train accuracy score: 0.90525
Test accuracy score: 0.8914166666666666
Train recall score: 0.9807312252964426
Test recall score: 0.8451834862385321
Train AUC score: 0.9400499366395979
Test AUC score: 0.8701115130689425
```

The modification of feature campaign improved the performance in both test precision and test recall with respect to the previous result.
The results with the original data show an higher accuracy, but a lower recall.
Furthermore, now the features are more interpretable.

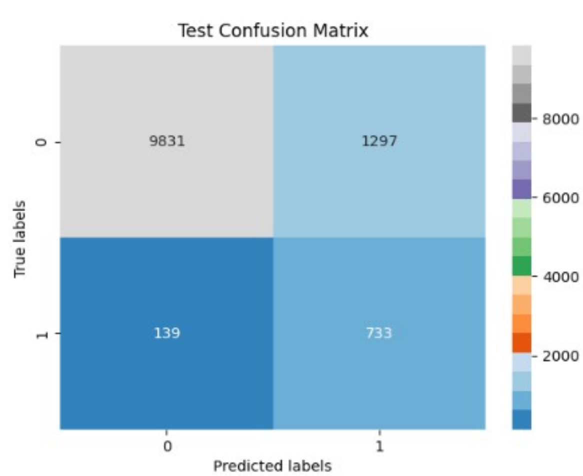


Once again we observe that the model picks up the trend

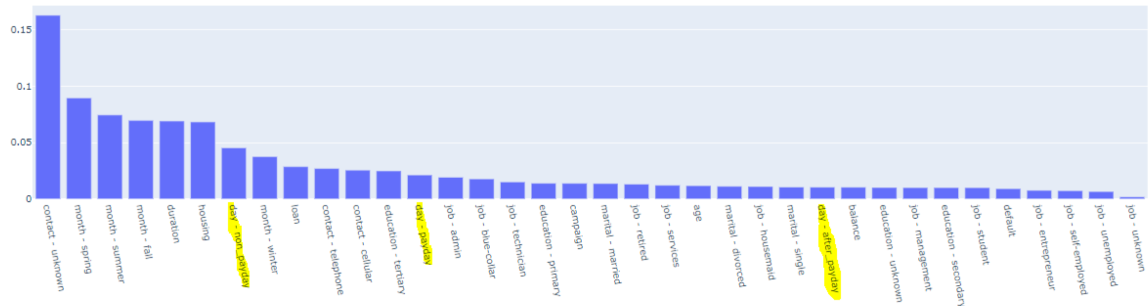
GROUP FEATURE DAY
Let's group feature day in three categories:
- payday (1, 10, 22, 30, 31)
- The 3 days after paydays
- The other days



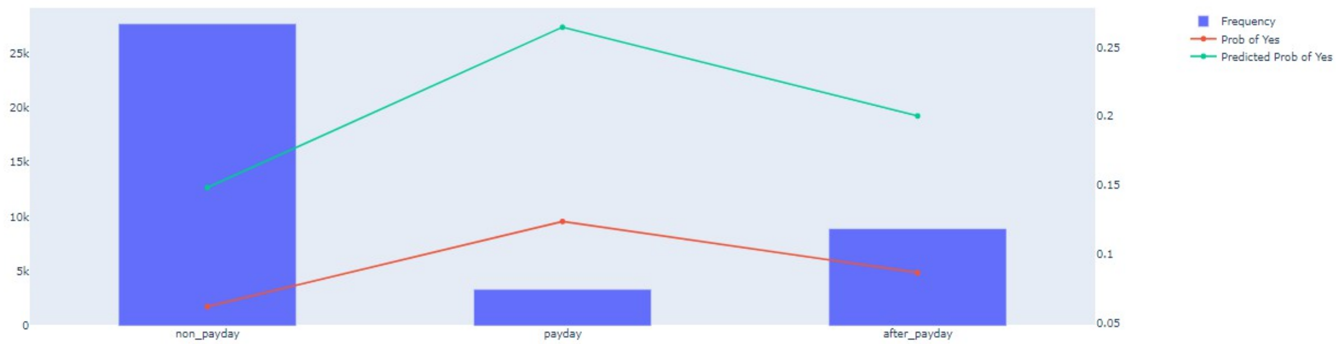
Once again, we can see a clear trend in the data



```
Train accuracy score: 0.8927857142857143
Test accuracy score: 0.8803333333333333
Train recall score: 0.9718379446640316
Test recall score: 0.8405963302752294
Train AUC score: 0.929232030539592
Test AUC score: 0.8620217452957744
```



The accuracy **reduced**, while the recall remained more or less on the same level

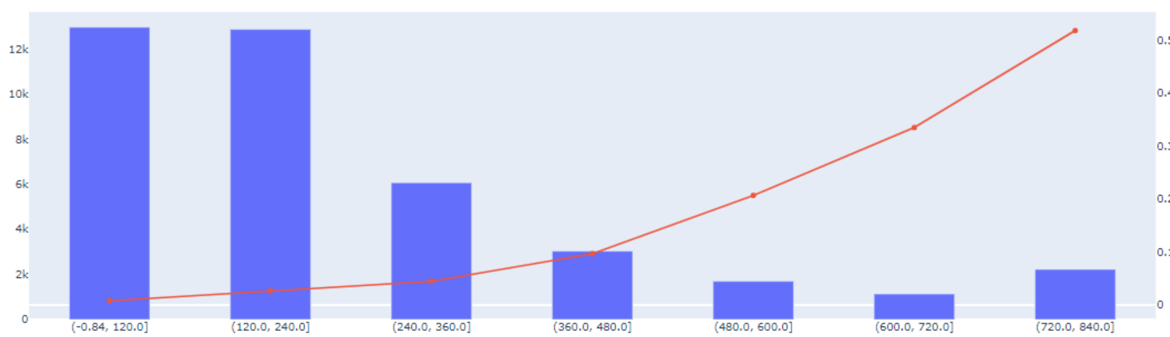


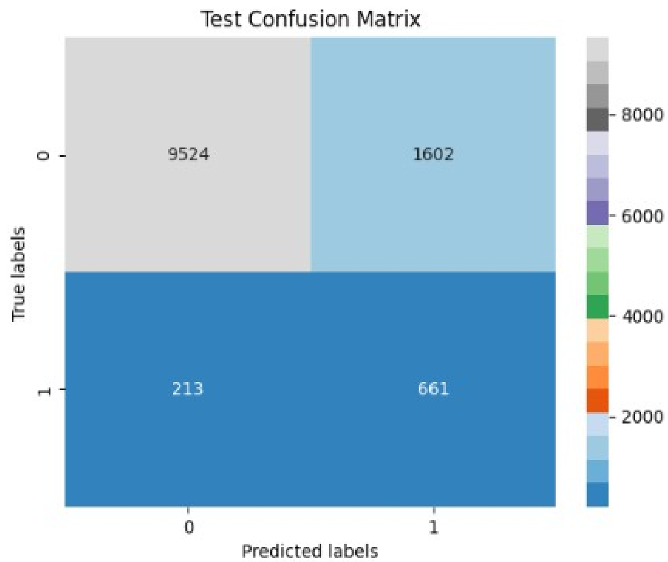
Once again the model picked up the trend

GROUP FEATURE DURATION

Based on the outlier analysis, we group together all the samples with duration longer than 14 minutes, and then we bin the values in groups of 2 minutes (originally it was 15, so 900 seconds, but 14 is even so divisible by two).

The result show a very clear trend in the probability of Yes

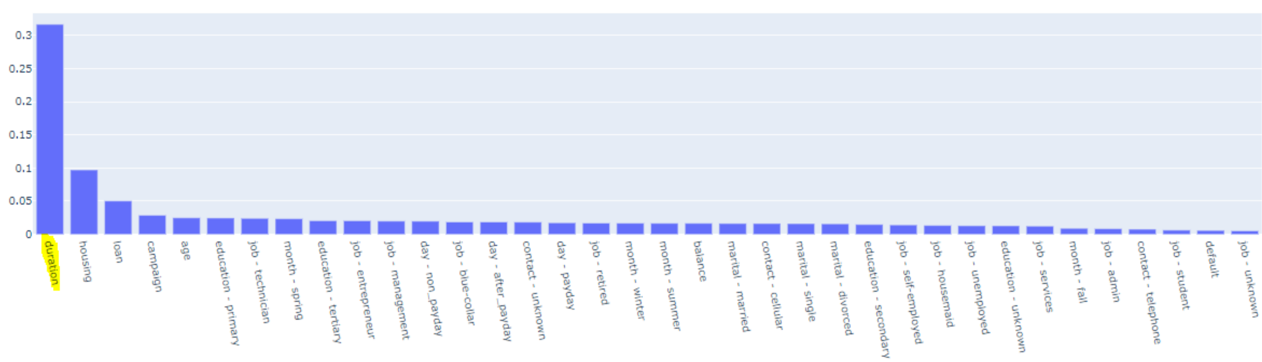




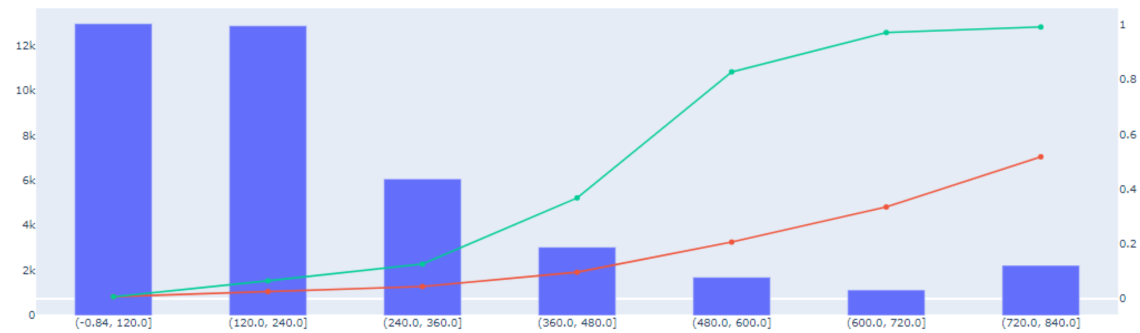
The performance **reduced quite a lot**

```
Train accuracy score: 0.8720714285714286
Test accuracy score: 0.84875
Train recall score: 0.9183976261127597
Test recall score: 0.7562929061784897
Train AUC score: 0.8934316254360857
Test AUC score: 0.8061529244176647
```

However, if we evaluate with 5 fold CV, the Test accuracy increases **from 54% to 68%** This indicates that the feature engineering for the duration is **effective** and it reduces the **variance** of the model.



The importance of feature duration increased more than 3x.

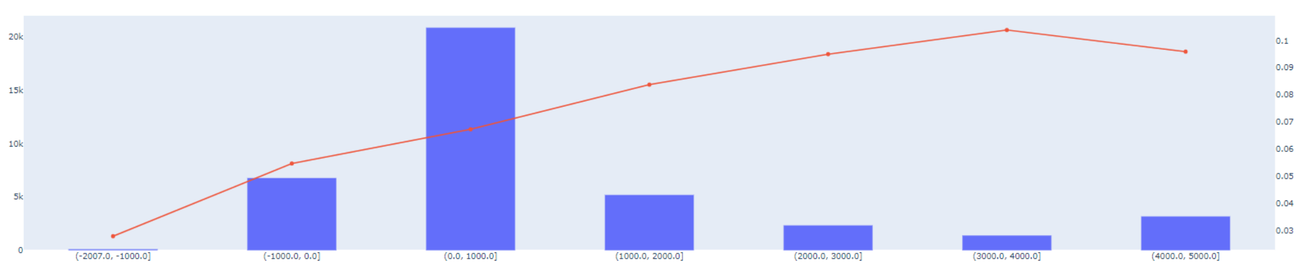


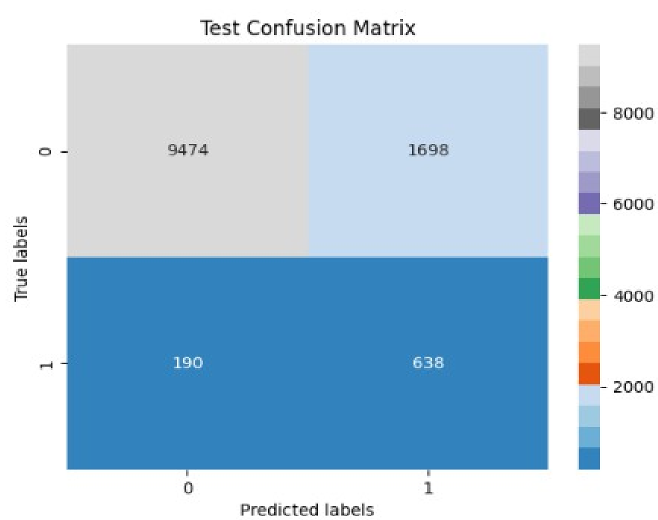
The observed vs predicted plot show that the model picks up the trend, but weights too much the long durations.
Most likely, it is overfitting on the longest durations.

It is still a good feature engineering, however we need to be mindful of the overfitting in the tuning phase

GROUP FEATURE BALANCE
By binning the feature balance, we reduce the number of possible values.
The operation results in a clear trend in the data

Frequency plot for attribute balance and probability of Yes for each category

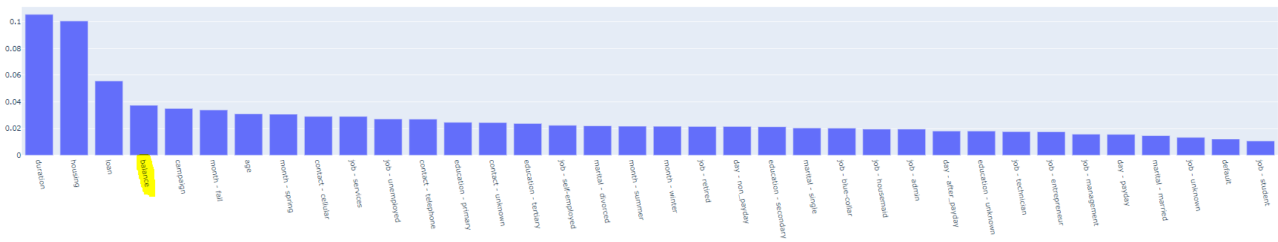




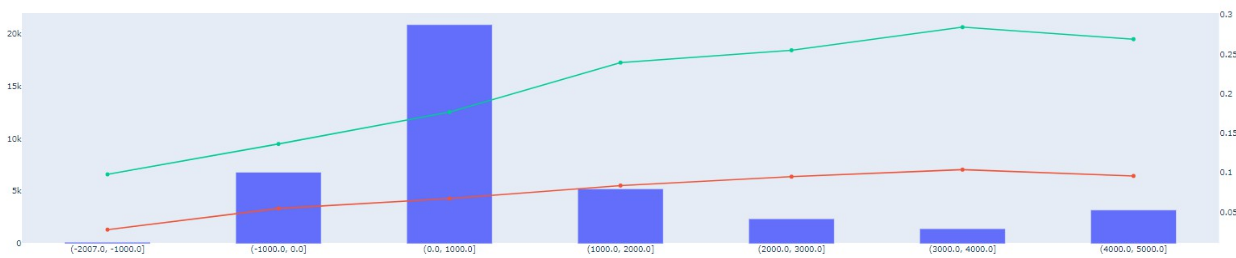

```
Train accuracy score: 0.8612142857142857
Test accuracy score: 0.8426666666666667
Train recall score: 0.905705996131528
```

```
Test recall score: 0.7705314009661836
Train AUC score: 0.8816861000247337
Test AUC score: 0.8092721451662281
```

The performed feature engineering slightly improves the performance.

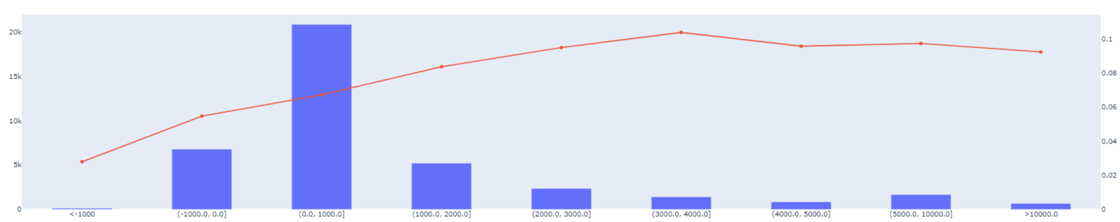


We can notice how the feature importance increased with the engineering.

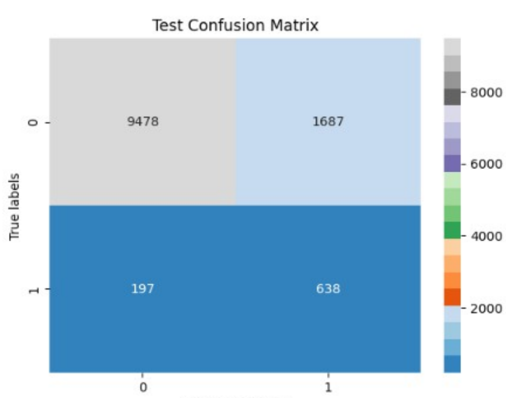


The model picks up the trend, but once again it is biased toward positive answer.
Also in this case, the feature engineering improves the 5-fold cross validation accuracy, going from 68% to 71%.

We try to expand the last bin: from >4000 to (4000, 5000], (5000, 10000], >10000

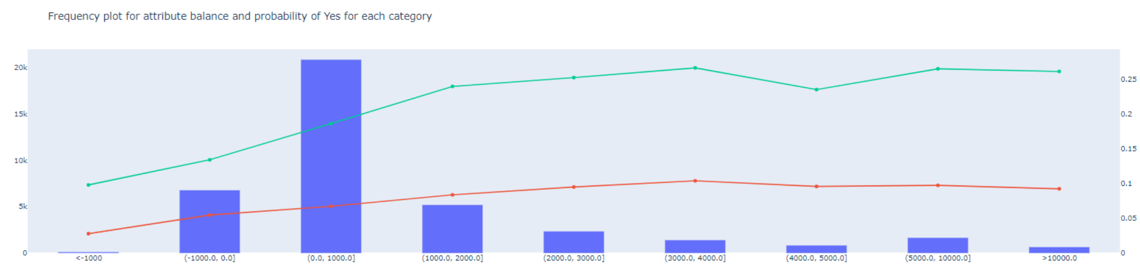


This change causes a slight drop of performance



```
Train accuracy score: 0.8625714285714285
Test accuracy score: 0.843
Train recall score: 0.9063561377971858
Test recall score: 0.7640718562874251
Train AUC score: 0.8827243120074253
Test AUC score: 0.8064873388020198
```

And indeed the model does not fully pick up the trend in the data:



Hence, we are going to keep the old binning

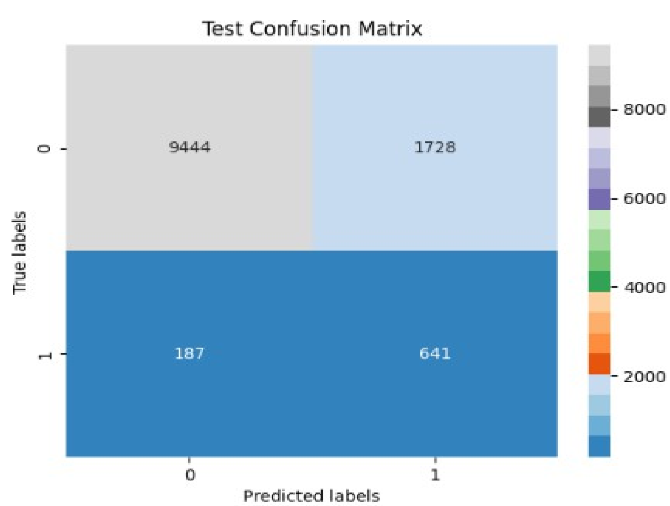
GROUP FEATURE JOB

Grouping:

'housemaid': 'low skills',
'blue-collar': 'low skills',

'admin': 'business jobs',
'services': 'business jobs',
'entrepreneur': 'business jobs',

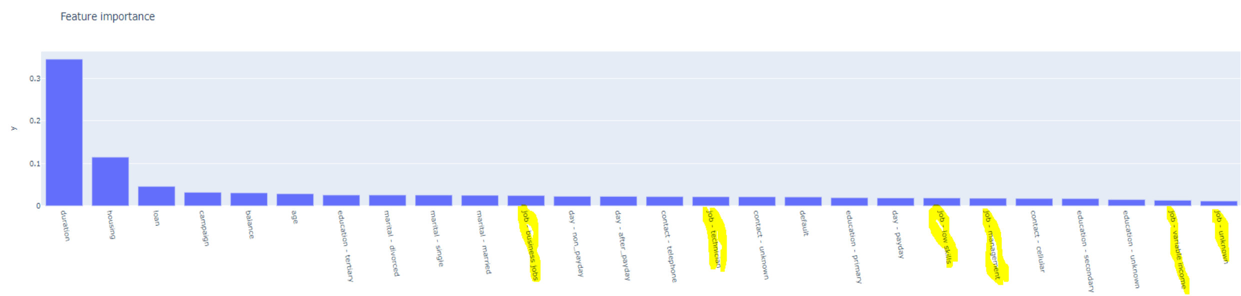
'self-employed': 'variable income',
'retired': 'variable income',
'unemployed': 'variable income',
'student': 'variable income',



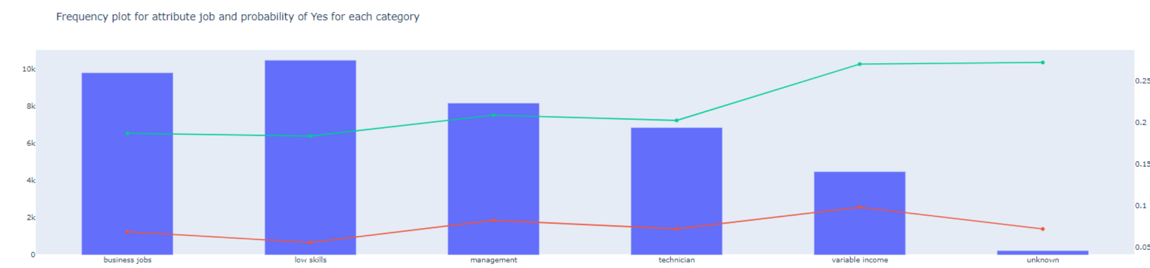

```
Train accuracy score: 0.8625714285714285
```

```
Test accuracy score: 0.843
Train recall score: 0.9063561377971858
Test recall score: 0.7640718562874251
Train AUC score: 0.8827243120074253
Test AUC score: 0.8064873388020198
```

Performance comparable with the ones without engineering



The feature importance show low relevance of the Job, but it did not change with the engineering



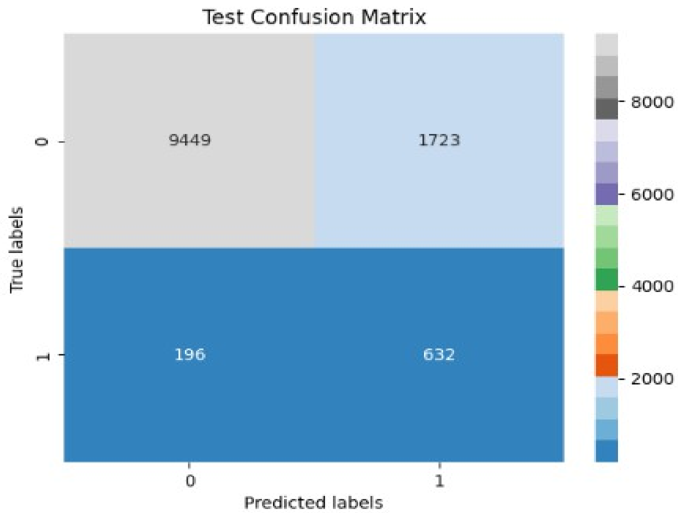
The model picks up the trend, except for the missing data:
probably this is due to the low number of samples.

Different grouping:
'housemaid': 'low skills',
'blue-collar': 'low skills',

'admin': 'business jobs',
'services': 'business jobs',

'entrepreneur': 'own boss',
'self-employed': 'own boss',

'retired': 'variable income',
'unemployed': 'variable income',
'student': 'variable income',




```
Train accuracy score: 0.8548928571428571
Test accuracy score: 0.8400833333333333
Train recall score: 0.8921663442940039
Test recall score: 0.7632850241545893
```

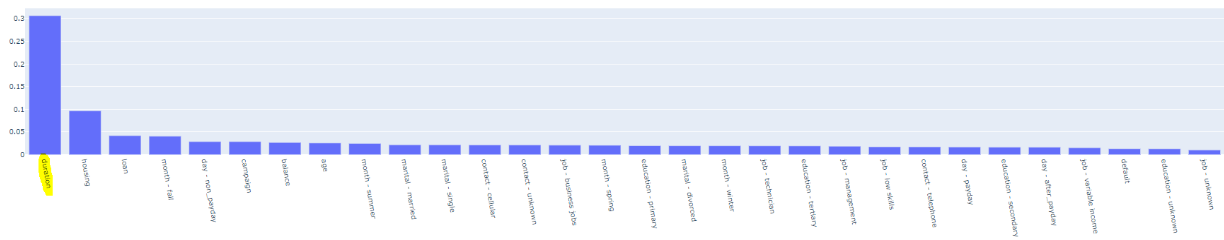
Performance slightly reduces. We are going to stick with the old grouping

PARAMETERS TUNING

We use the RandomSearchCV to perform the tuning.
Parameters used so far

```
min_child_weight=10,
max_depth=4,
n_estimators=200,
scale_pos_weight=spw        # 12.812
```

```
Train accuracy score: 0.8581071428571428
Test accuracy score: 0.8405
Train recall score: 0.9013539651837524
Test recall score: 0.7729468599033816
Train AUC score: 0.8780061511866625
Test AUC score: 0.809226741802747
```



The feature importance is highly unbalance on the **duration**, which is a feature that **cannot be controlled ex-ante**.

 # Iteration 1

```
param_grid = {'max_depth': [2, 4, 6, 8],            # default 6
              'subsample': [0.95, 1],               # default 1
              'colsample_bytree': [0.5, 0.7, 0.9, 1],  # default 1
              'min_child_weight': [5, 10, 20, 50],     # default 1
              'n_estimators': [200, 500, 1000],      # default 100
              'learning_rate': [0.01, 0.1, 0.3],    # default 0.3
              'scale_pos_weight': [8, spw, 20],
              'random_state': [42]}
```

Top 5:

max_depth	subsample	colsample_bytree	min_child_weight	n_estimators	learning_rate	scale_pos_weight	test_score	train_score
8	0.95	0.5	5	1000	0.3	20	0.899679	0.994250
8	1	0.7	10	500	0.1	8	0.891571	0.956161
2	0.95	0.7	5	200	0.01	8	621.58.00	0.889500
8	1	0.9	20	200	0.3	8	0.886643	0.947384
8	0.95	0.9	20	1000	0.1	20	0.883536	0.952830

 # Iteration 2

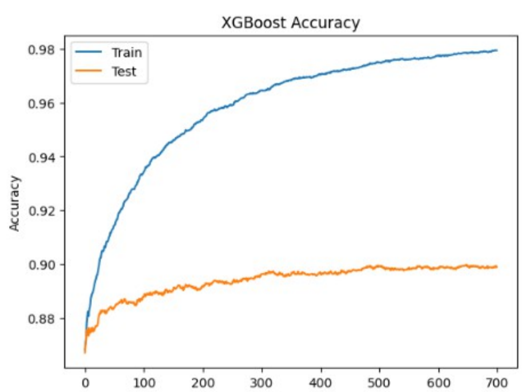
```
param_grid = {'reg_lambda': [0, 1, 5],
        'reg_alpha': [0, 1, 5],
        'max_depth': [6, 8, 10],
        'subsample': [0.95, 1],
        'colsample_bytree': [0.5, 0.7],
        'min_child_weight': [5, 10],
        'n_estimators': [500, 700],
```

'learning_rate': [0.01, 0.1, 0.3],
'scale_pos_weight': [8, spw, 20],
'random_state': [42]}

Top 5

lambda	alpha	max_depth	subsample	colsample_bytree	min_child_weight	n_estimators	l_rate	s_p_w	test_score	train_score
0	0	8	0.95	0.7	10	700	0.3	8	0.899964	0.990545
0	0	8	0.95	0.5	5	500	0.1	8	0.899821	0.971438
0	0	8	0.95	0.5	5	700	0.1	12.812155	0.895393	0.974098
1	5	6	1	0.5	5	700	0.3	8	0.885893	0.937446
0	1	10	1	0.7	10	700	0.01	8	0.884393	0.918768

We use the best performing combination:
opt_params = {'reg_lambda': 0,
'reg_alpha': 0,
'max_depth': 8,
'subsample': 0.95,
'colsample_bytree': 0.7,
'min_child_weight': 10,
'n_estimators': 700,
'learning_rate': 0.3,
'scale_pos_weight': 8,
'random_state': 42}



We are not overfitting, and we can see the model learning.

```
Train accuracy score: 0.9794285714285714
Test accuracy score: 0.8988333333333334
Train recall score: 0.9956479690522244
Test recall score: 0.4178743961352657
Train AUC score: 0.9868915458403186
Test AUC score: 0.6761767254575363
```

The accuracy is good, but the test recall is very low:
the model is learning mainly to recognize the negatives.
Our goal, however, is to classify correcly mainly the positive, so we are going to change the Randomized **Search objective** from accuracy to **recall**.

Note that we will **not include the scale pos weight** parameter in the search, because otherwise the model will mainly lean on it to improve the recall.
We decide to fix it at the suggested value # neg / # pos

 # Iteration 1

```
param_grid = {'max_depth': [2, 4, 6, 8],             # default 6
          'subsample': [0.95, 1],             # default 1
          'colsample_bytree': [0.5, 0.7, 0.9, 1],   # default 1
          'min_child_weight': [5, 10, 20, 50],      # default 1
          'n_estimators': [200, 500, 1000],      # default 100
          'learning_rate': [0.01, 0.1, 0.3],    # default 0.3
          'scale_pos_weight': [spw],
          'random_state': [42]}
```

Top 3:

max_depth	subsample	colsample_bytree	min_child_weight	n_estimators	learning_rate	scale_pos_weight	test_score	train_score
6	0.95	1	10	500	0.01	12.812155	0.793540	0.860494
8	0.95	0.5	50	1000	0.01	12.812155	0.773711	0.875363
8	1	0.7	5	200	0.01	12.812155	0.764527	0.877055

Iteration #2

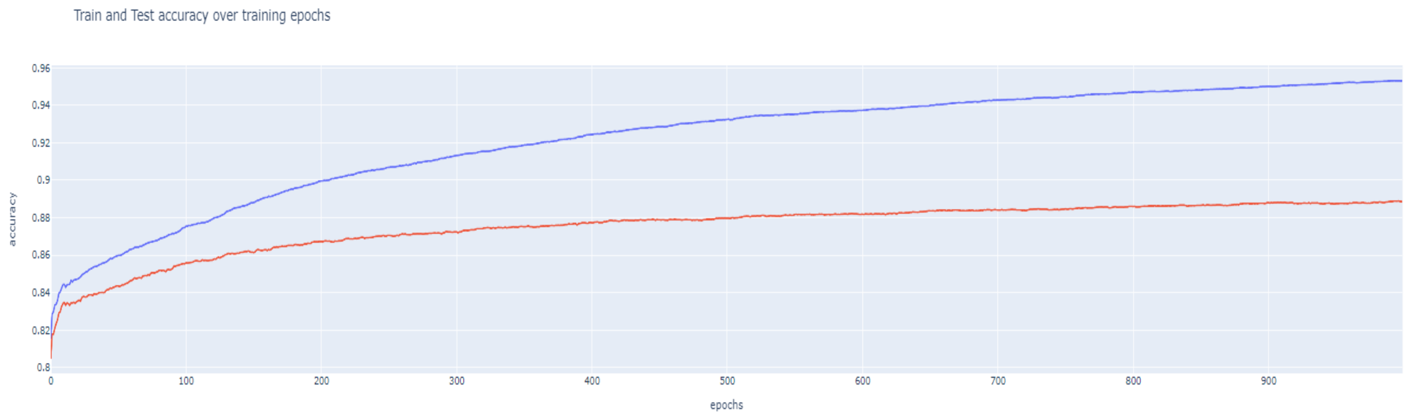
```
param_grid = {'reg_lambda': [0, 1, 5],
          'reg_alpha': [0, 1, 5],
          'max_depth': [6, 8, 10],
          'subsample': [0.8, 0.95, 1],
          'colsample_bytree': [0.7, 0.9, 1],
          'min_child_weight': [10, 20, 50],
          'n_estimators': [500, 700, 1000],
          'learning_rate': [0.01, 0.02, 0.05],
          'scale_pos_weight': [spw],
          'random_state': [42]}
```

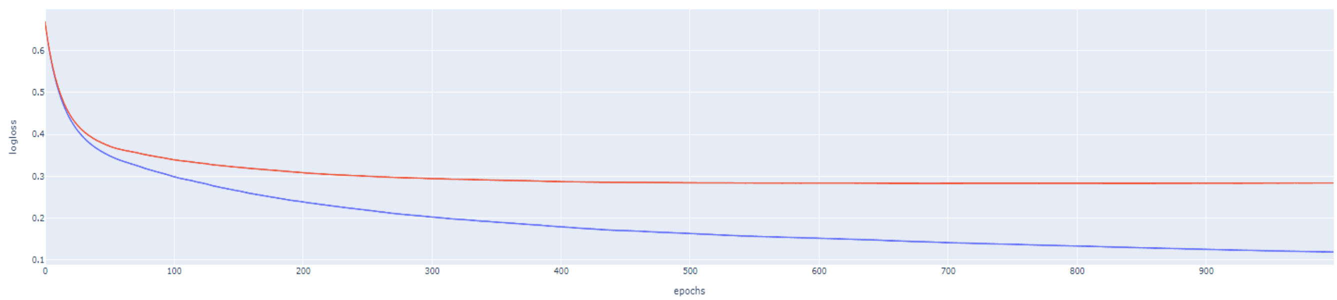
Top 3:

lambda	alpha	max_depth	subsample	colsample_ bytree	min_child_ weight	n_estimators	l_rate	s_p_w	test_score	train_score
5	0	10	1	1	10	1000	0.05	12.812155	0.886179	0.958286
5	0	8	0.95	1	10	1000	0.05	12.812155	493.59.00	0.947241
0	5	8	0.95	0.9	10	700	0.05	12.812155	0.872607	0.923893

We use the best performing tuning

```
opt_params = {'reg_lambda': 5,
          'reg_alpha': 0,
          'max_depth': 10,
          'subsample': 1,
          'colsample_bytree': 1,
          'min_child_weight': 10,
          'n_estimators': 1000,
          'learning_rate': 0.05,
          'scale_pos_weight': spw,
          'random_state': 42}
```



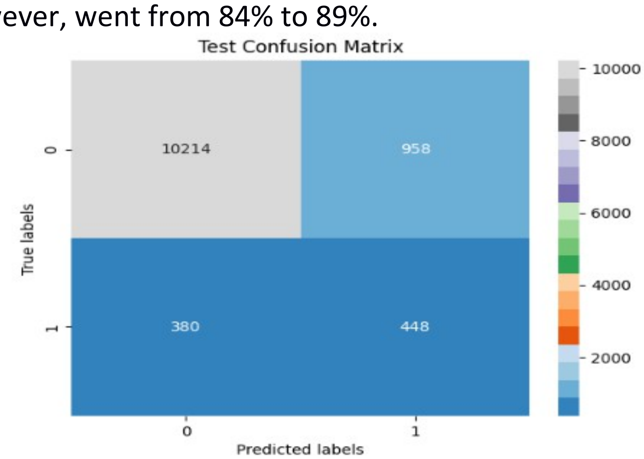


The Test Negative Log-likelihood is monotonic decreasing:
better than the previous tuning in which it was increasing after a bit

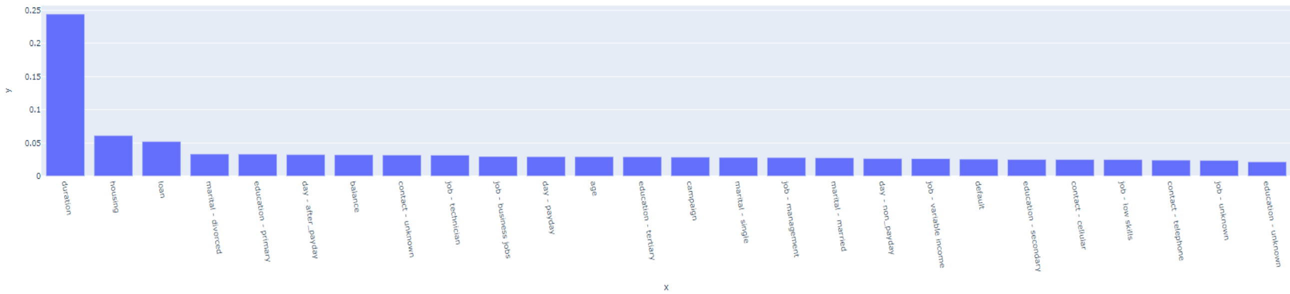
```
Train accuracy score: 0.953
Test accuracy score: 0.8885
Train recall score: 0.9970986460348162
Test recall score: 0.5410628019323671
Train AUC score: 0.9732909549779202
Test AUC score: 0.7276563562114395
```

The accuracy is comparable to the **previous tuning**, but the Recall is 15% higher.

It's weird the fact that the Randomized Search was made to maximise Recall, but the Test Recall dropped from 77% to 54% with respect to the **initial tuning**.
The accuracy, however, went from 84% to 89%.



The number of False Negative almost doubled.



Duration is dominating the feature importance (23%) but better than without tuning (30%).

5-fold cross validation
Initial tuning:

```
Train accuracy score: 0.8549000000000001
Test accuracy score: 0.7263249999999999
Train recall score: 0.886308638634766
Test recall score: 0.6939550949913643
```

Tuned model:

```
Train accuracy score: 0.95059375
Test accuracy score: 0.761125
Train recall score: 0.9962013517270785
Test recall score: 0.5454231433506045
```

as expected, also the 5 fold cross validation shows that the tuning improved the test accuracy (+4%) at the cost of the recall (-15%).

We are going to keep the **initial tuning value**:
min_child_weight=10,
max_depth=4,
n_estimators=200,
scale_pos_weight=spw # 12.812

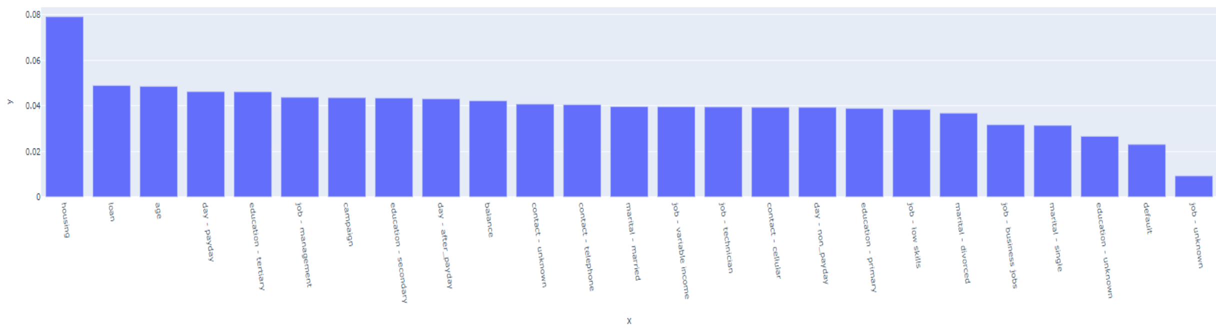
Drop feature duration

We try to drop the feature duration:
it is by far the most important feature, however it cannot be controlled and it is unknown prior the call.

Initial tuning

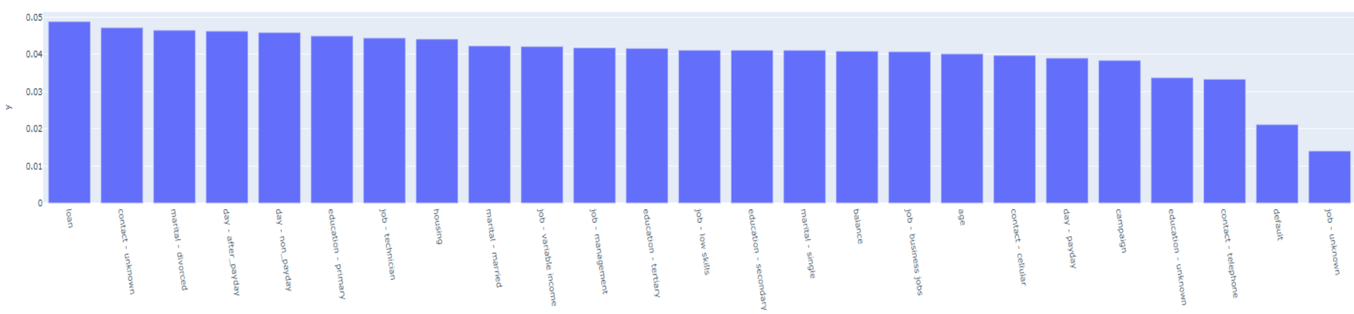
```
Train accuracy score: 0.69475
Test accuracy score: 0.6605833333333333
Train recall score: 0.7253384912959381
Test recall score: 0.43478260869565216
```

The results are worse than before, but the feature importance is more homogeneous



Tuned model

```
Train accuracy score: 0.86775
Test accuracy score: 0.77825
Train recall score: 0.9680851063829787
Test recall score: 0.2391304347826087
```



Feature importance is very distributed.

All considered, we still should **keep** the feature Duration as it contains important information