In [1]:
```python
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
from eda_utils import split_dataset, run_training, \
                        dist_by_labels, compute_metric_on_test
```

Dataset analysis

I will start by looking at the data, checking labels distributions and searching for correlations betbeeen features and target label.

The dataset appear to be very clean, and clearly synthetic, each feature is well distributed and seems to be uncorrelated to the labels provided.

In [2]:
```python
# Load
df_sales = pd.read_csv('sales.csv')
df_ca = pd.read_csv('customer_activity.csv')
df_cal = pd.read_csv('customer_activity_latest.csv')
df_labels = pd.read_csv('labels.csv')
df_stores = pd.read_csv('stores.csv')
df_cp = pd.read_csv('customer_profiles.csv')
df_ft = pd.read_csv('feature_store.csv')
```

CUSTOMER ACTIVITY - CLARIFY

In [3]:
```python
# what is the difference between CA and CA latest?
all_scd_a = set(df_cal['scd_a'])
df_cao = df_ca[~df_ca['scd_a'].apply(lambda x: x in all_scd_a)]

# old values have all a closing date?
len(df_cao) == len(df_ca) - len(df_cal)
```

Out[3]:
```
True
```

In [4]:
```python
# Transactions that have expired are not in the latest dataframe
display(Counter(df_cao['valid_to'].isna()))
display(Counter(df_cal['valid_to'].isna()))
display(Counter(df_ca['valid_to'].isna()))
```

```
Counter({False: 16617})
Counter({True: 150000})
Counter({True: 150000, False: 16617})
```

CUSTOMER ACTIVITY ATTR

In [5]:
```python
# CA in CAL are unique (TOP value counts are 1)
df_cal['idx'].value_counts().head(3)
```

Out[5]:
```
0         1
100012    1
99996     1
Name: idx, dtype: int64
```

In [6]:
```python
# ASSUMPTION: all the idx are unique to the customers
# add attr columns from CAL
for col in ['scd_a', 'scd_b']:
    df_cp[col] = df_cp['idx'].map(df_cal.set_index('idx')[col])
```

CUSTOMER PROFILES GLOBAL ANALYSIS

Here we will see that the features are simply a random sample over precise ratios. It is a mock dataset after all!

In [7]:
```python
# setup categorical features
cat_features = ['attr_a', 'attr_b', 'attr_c', 'scd_b']
```

In [8]:
```python
# "attr_a" attribute is sampled randomly with ratios [.5, .2, .15, .1, .05]
df_cp['attr_a'].value_counts() / len(df_cp)
```

Out[8]:
```
1    0.502120
5    0.198253
4    0.148927
3    0.100387
2    0.050313
Name: attr_a, dtype: float64
```

In [9]:
```python
# "attr_b" attribute is sampled randomly with ratios [.3, .3, .2, .1]
df_cp['attr_b'].value_counts() / len(df_cp)
```

Out[9]:
```
d    0.398387
c    0.300553
b    0.200113
a    0.100947
Name: attr_b, dtype: float64
```

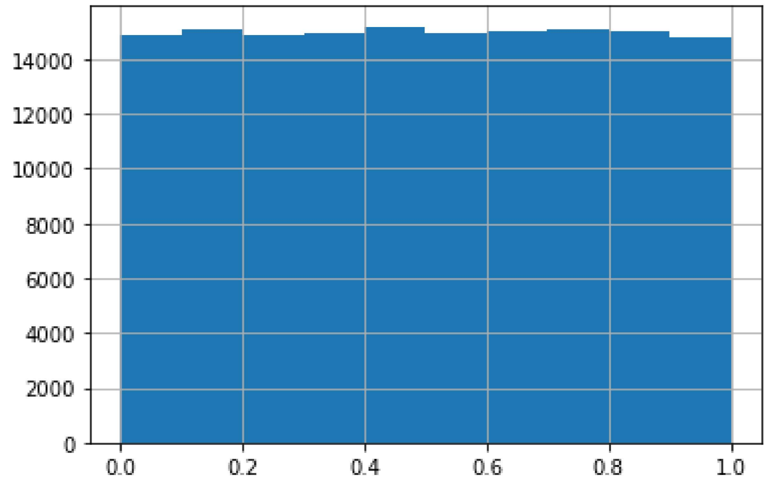
In [10]:
```python
# "attr_c" attribute is sampled randomly with ratios [2/3, 1/3]
df_cp['attr_c'].value_counts() / len(df_cp)
```

Out[10]:
```
yes    0.6667
no     0.3333
Name: attr_c, dtype: float64
```

In [11]:
```python
# "scd_b" attribute is sampled randomly with ratios .2 for all variables
df_cp['scd_b'].value_counts() / len(df_cp)
```

Out[11]:
```
5    0.200753
2    0.200540
3    0.200380
1    0.199327
4    0.199000
Name: scd_b, dtype: float64
```

In [12]:
```python
# "scd_a" attribute is sampled uniformely from the interval [0, 1]
df_cp['scd_a'].hist()
plt.show()
```



ADD LABELS

In [13]:
```
# add label column to df
df_cp['label'] = df_cp['idx'].map(df_labels.set_index('idx')['label'])
```

CHECK DATA IS CLEARN

In [14]:
```
df_cp.info()
```

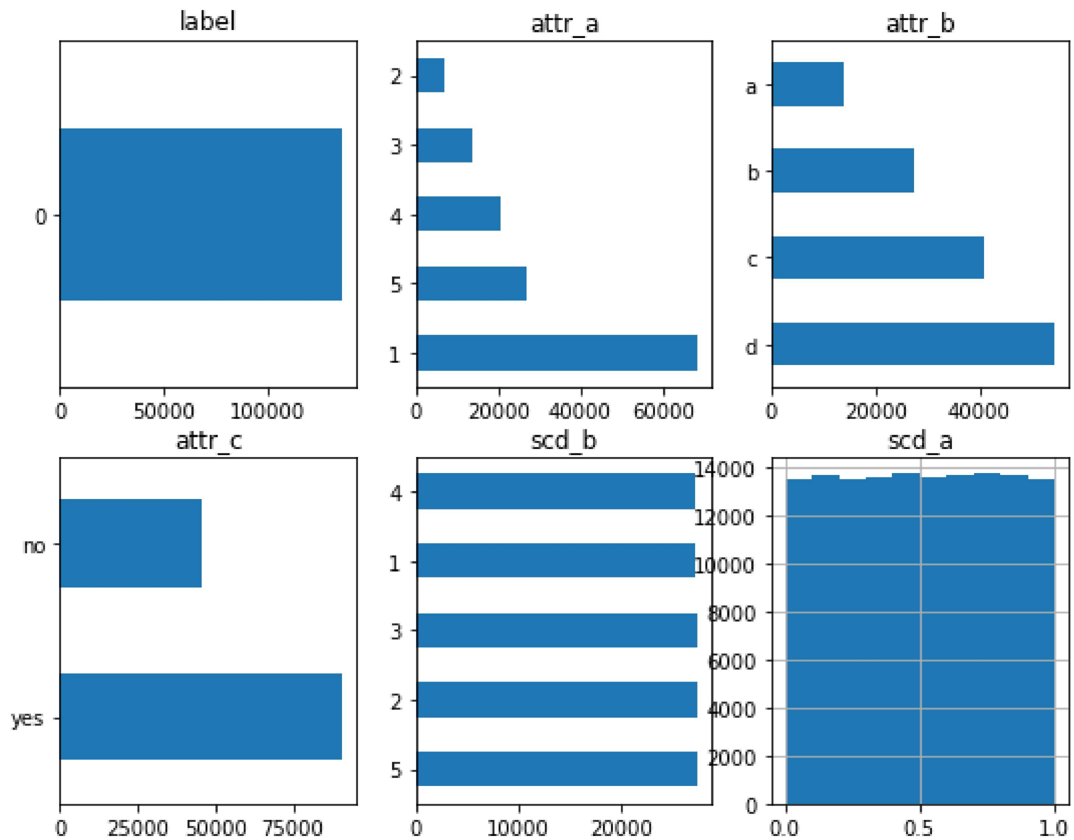
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   idx     150000 non-null  int64  
 1   attr_a  150000 non-null  int64  
 2   attr_b  150000 non-null  object 
 3   attr_c  150000 non-null  object 
 4   scd_a   150000 non-null  float64
 5   scd_b   150000 non-null  int64  
 6   label   150000 non-null  int64  
dtypes: float64(1), int64(4), object(2)
memory usage: 8.0+ MB
```

CHECK DISTRIBUTIONS BY LABELS

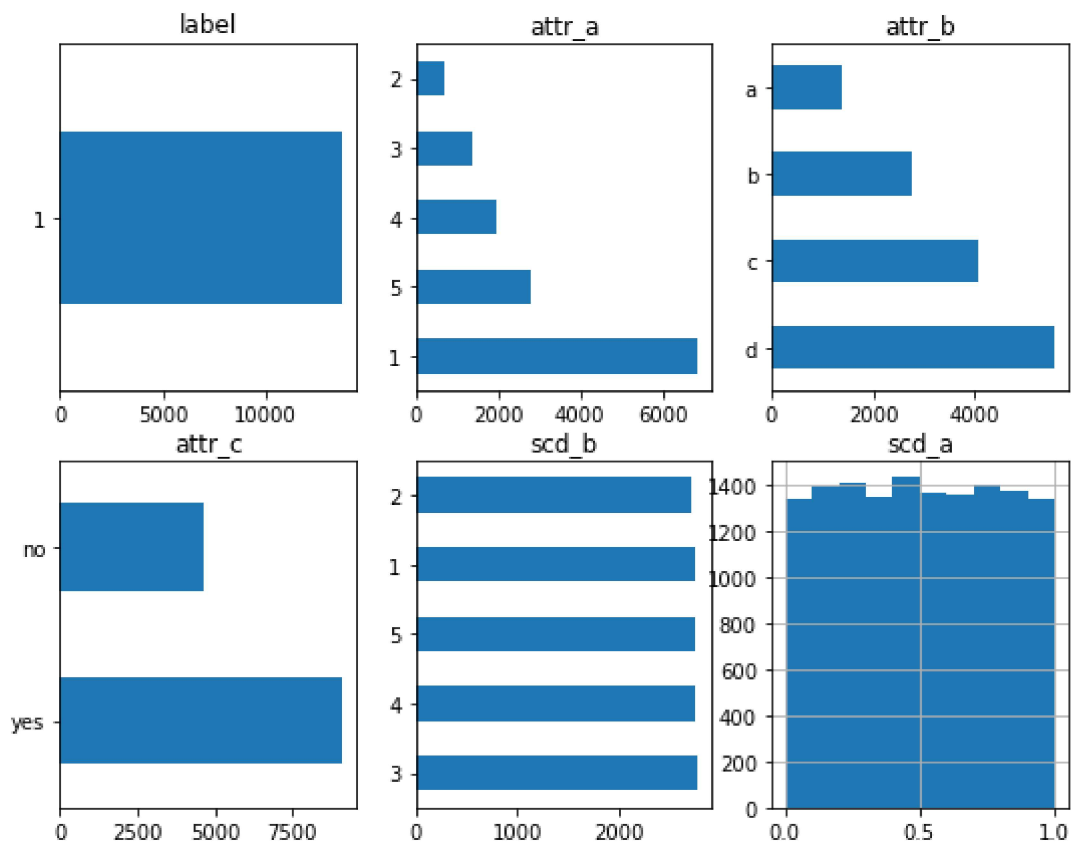
Features are perfectly distributed over each label. This tell us that there is not going to be any correlation, or pattern that we can extract from these features to predict the labels.

In [15]:
```
dist_by_labels(df_cp, cat_features)
```

```
------------ LABEL 0 -----------
```




```
------------ LABEL 1 -----------
```



CHECKING for label distribution for CA not in CAL

The label distribution continue to be at identically distributed, with a ratio 1/10

In [16]:
```
df_cp.loc[df_cao['idx'].values]['label'].value_counts()
```

Out[16]:
```
0    15074
1     1543
Name: label, dtype: int64
```

Discussion

The labels are clearly randomly sampled form a dataset with fixed distribution with ration 1 / 10. All the attributes are distributed identically for both labels, so they won't be able to discriminate the label without overfitting to the dataset at hand.

Nevertheless, for sake of providing some methodology, here I provide an idea of how I would approach a real case scenario :

- Split the dataset in 3, training, testing and validation (or K-folds)
- the testing dataset will not be used in the model selection, as it will be our bias-free dataset to have a "production like" metric to hold.
- preprocess dataset, mapping categorical to ints or one-hot encoded features
- Train distinct model structures (NB, trees, ...)

Once the model is selected:

- Run a feature selection algorithm to find out which features are helping the performance of the model
- Run hyperparameter search for selecting the optimal model setup

Preprocessing

In [17]:
```
# init
df = df_cp.copy()
all_features = cat_features + ['scd_a']

# make up feature maps
unique_values = df['attr_b'].unique()
feature_map = dict(zip(sorted(unique_values), range(len(unique_values))))

# map cat features
df['attr_b'] = df['attr_b'].replace(feature_map)
df['attr_c'] = df['attr_c'].replace({'yes': 1, 'no': 0})

# final setup
df.head(3)
```

Out[17]:

	idx	attr_a	attr_b	attr_c	scd_a	scd_b	label
0	0	3	2	1	0.104834	2	0
1	1	4	3	1	0.434486	2	0
2	2	5	3	1	0.094773	1	1

Model selection

Here we will split the dataset. Moreover because the labels are skewed, I am going to downsample the majority class and train models on a balanced dataset.

In [18]:
```
# store away the test set
X, X_test, y, y_test = split_dataset(df, all_features)
```

In [19]:
```
# training df
df_train = pd.DataFrame(X)
df_train['label'] = y

# downsample
df_neg = df_train[df_train['label']==0]
df_pos = df_train[df_train['label']==1].reset_index(drop=True)
df_neg_down = df_neg.sample(len(df_pos), random_state=42).reset_index(drop=True)

# join back
df_train = (
    pd.concat([df_neg_down, df_pos])
    .sample(frac=1, random_state=42)
    .reset_index(drop=True)
)
X = df_train.drop(columns=['label']).values
y = df_train['label'].values
```

In [20]:
```
# run models and check performance
best_model, best_metric = run_training(X, y)
```

```
Model: Nearest Neighbors
Score: 0.251518095700753
Inference time: 0.21451703707377115

Model: Decision Tree
Score: 0.15359080236418104
Inference time: 0.0015673637390136719

Model: Random Forest
Score: 0.20727066634280625
Inference time: 0.006380001703898112

Model: Neural Net
Score: 0.15391466278034166
Inference time: 0.011669317881266275

Model: AdaBoost
Score: 0.264310582139098
Inference time: 0.040102640787760414

Model: Naive Bayes
Score: 0.24876528216338759
Inference time: 0.0009929339090983074

Model: QDA
Score: 0.24641729414622296
Inference time: 0.0012861887613932292

Model: GBoost
Score: 0.23844223139826734
Inference time: 0.00558622678120931
```

In [21]:

```
# best model found
display(best_model)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('adaboostclassifier', AdaBoostClassifier())])
```

In [22]:

```
# compute metrics on test set
compute_metric_on_test(best_model, X, y, X_test, y_test)
```

Out[22]:

	accuracy	precision	recall	f_score
0	0.657467	0.027133	0.027133	0.027133

Picked model

As expected, the models are having a hard time improving on the metric, as they do not find meaningful correlation between attributes and labels. The best performing model seems to be AdaBoostClassifier.

If multiple models would compete for the first place, I would use the inference time as a decision point, as it would be important if this model will be used for APIs.

Overfitting possibilities

We could now look at the hyperparameters for the model and run a grid search for the optimal setup. We can expect only minor inprovements in performance, with the risk of picking parameters that overfit the specific validation test at hand (althogh the k-fold should help against that).

We could also use *Permutation importance* to select a subset of the features used. Considering the dataset at hand has only 5 features, it is not needed to make feature selection part of our pipeline.