Starbucks Capstone Project

1. Definition

1.1. Project Overview

The data set contains simulated data that mimics customer behavior on the Starbucks rewards mobile app. Once every few days, Starbucks sends out an offer to users of the mobile app. An offer can be:

- Merely an ***advertisement*** for a drink or an actual offer such as a ***discount***;
- ***BOGO*** (buy one get one free);
- Some users might ***not receive*** any offer during certain weeks.

Not all users receive the same offer, and that is the challenge to solve with this data set in order to improve the customer journey.

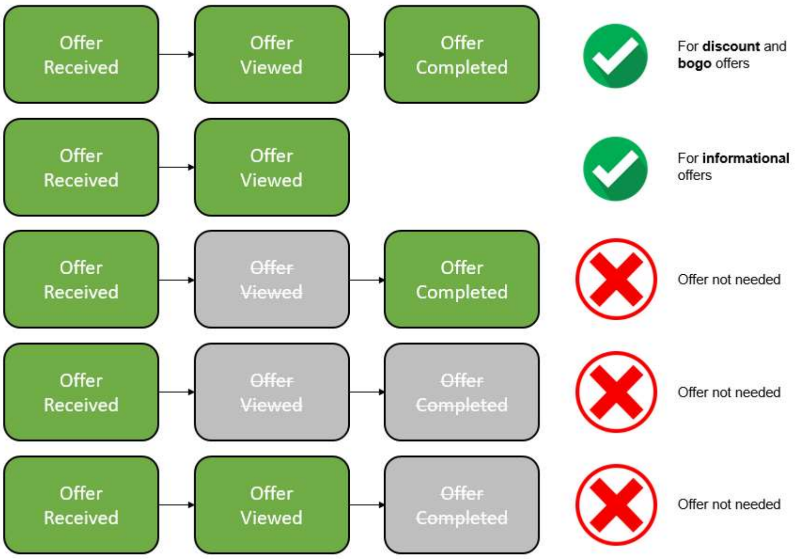
1.2. Problem Statement

A key challenge too, it is not only discovering which one is the best offer to be sent, sometimes the person is static to it, being the goal to discover whether a client will buy in or not the offer. A group of individuals are few times assiduous customers, but even receiving the offer he or she does not use it, or do not even look at it, using it sometimes without realizing, only discovering the discount or the promo at the cashier.

The main goal of the offer is to avoid churn and keep clients around, when the churn probability increases, but if a customer is “uncorrelated” with offers and promotions, even being a premium client or not, does not add value to the business sending a promotion, since it has a cost aggregate to it.

A task within the project is to combine transaction, demographic and offer data to determine which demographic groups respond best to which offer type. This data set is a simplified version of the real Starbucks app because the underlying simulator only has one product whereas Starbucks actually sells dozens of products. Even gathering outside data if it is pertinent in order to achieve better performance for the model.

Figure 1. Possibilities of customer journey



With the data at hand, the following steps will be seek:

1. Doing an ETL (Extract, Transform and Load) of the data, converting its raw format as json;
2. Preparing the data as a pandas dataframe in order to be evaluated;
3. Performing an EDA (Exploratory Data Analysis) of the clean dataset to get insights;
4. Pre-processing of the data;
5. Tuning the model in order to find the best hyperparamenters;
6. Training the model;
7. Saving the model in order to develop an endpoint;
8. Developing a lambda function to invoke the API calling the endpoint.

1.3. Metrics

Few metrics will be evaluated in order to seek the best model, with the best parameters tuned. The main goal is to pursue the best **accuracy**, being aware of the **recall** and **precision** of the offer.

It is not wanted that an offer is only send to just a few people, or send to everybody, it is important to balance all the metrics (which is why **f1 score** is going to be looked too), since that can cause a very poor customer journey and increase churn, which means losing money.

Besides that, something that will be taken into consideration, are business rules, in order to see if a model is really needed. Because what is the purpose of spending money building an infrastructure to host a service, even if it cheap, if it cannot beat out simple if-else rules made the managers or similar.

Figure 2. Metrics to be evaluated

$$Precision = \frac{TP}{TP + FP}$$

$$Recall \ = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$F1\ Score = \frac{2 * (Precision + Recall)}{Precision + Recall}$$

TP	True positive
FP	False positive
TN	True negative
FN	False negative

2. Analysis

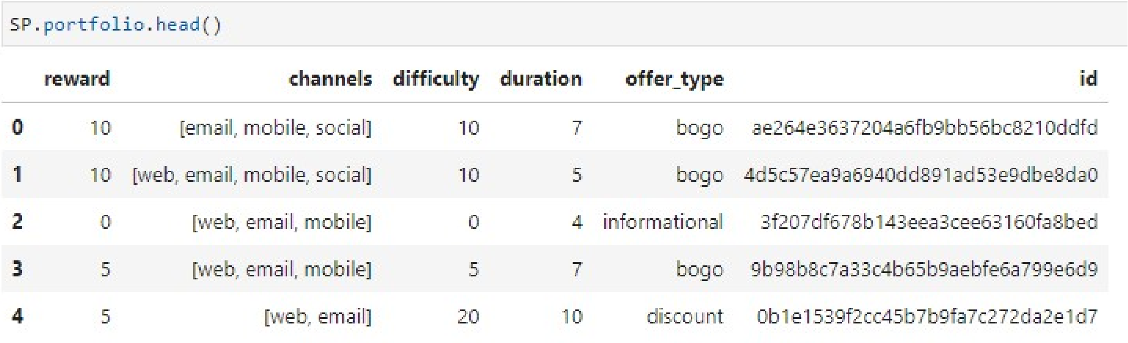
2.1. Data Exploration

Below it is possible to have a brief view of how the data is structure before the ETL is applied to it.

portfolio.json (10 x 6)

- id (string) - offer id
- offer_type (string) - type of offer ie BOGO, discount, informational
- difficulty (int) - minimum required spend to complete an offer
- reward (int) - reward given for completing an offer
- duration (int) - time for offer to be open, in days
- channels (list of strings)

Figure 3. Header of the portfolio data set



```
SP.portfolio.head()
```

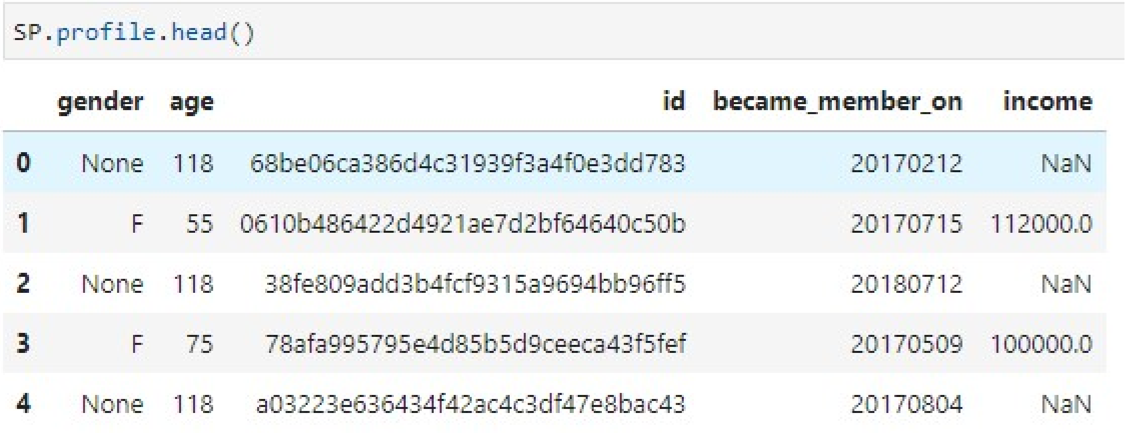
	reward	channels	difficulty	duration	offer_type	id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7

profile.json (17000 x 6)

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
- id (str) - customer id

- income (float) - customer's income

Figure 4. Header of the profile data set



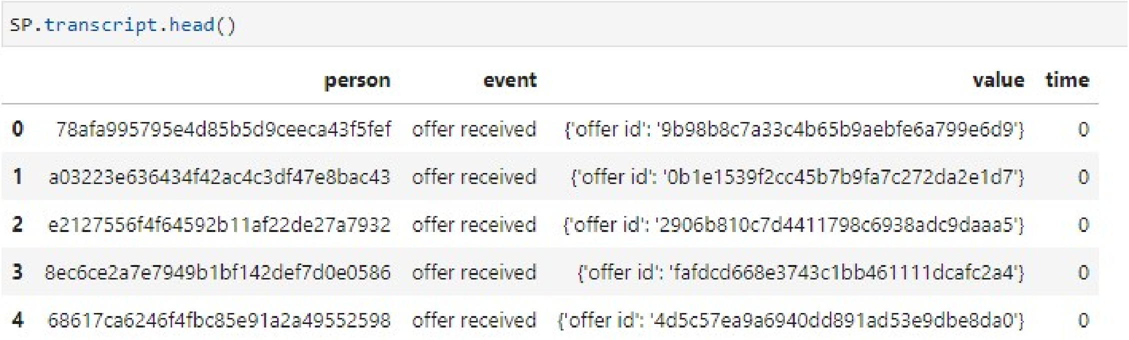
```
SP.profile.head()
```

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

transcript.json (306534 x 7)

- event (str) - record description (ie transaction, offer received, offer viewed, etc.)
- person (str) - customer id
- time (int) - time in hours since start of test. The data begins at time t=0
- value - (dict of strings) - either an offer id or transaction amount depending on the record

Figure 5. Header of the profile data set



```
SP.transcript.head()
```

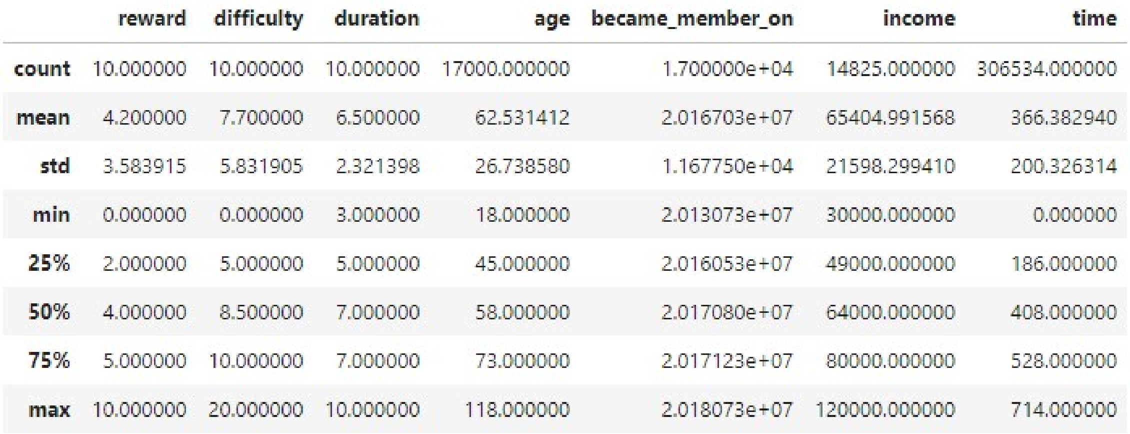
	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	0
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	0

The first basic treatments that were execute into the dataset were transforming it into full tabular data, avoiding lists and json inside a cell in the dataframe in order to be easier to work with it.

Furthermore, the field of membership was splited into year, month and day. Besides that, few fields' names were converted in order to be feasible to join all the tables.

As a first view of the summary from the dataset it is possible to extract the following table.

Figure 6. Summary statistics of the whole dataset



	reward	difficulty	duration	age	became_member_on	income	time
count	10.000000	10.000000	10.000000	17000.000000	1.700000e+04	14825.000000	306534.000000
mean	4.200000	7.700000	6.500000	62.531412	2.016703e+07	65404.991568	366.382940
std	3.583915	5.831905	2.321398	26.738580	1.167750e+04	21598.299410	200.326314
min	0.000000	0.000000	3.000000	18.000000	2.013073e+07	30000.000000	0.000000
25%	2.000000	5.000000	5.000000	45.000000	2.016053e+07	49000.000000	186.000000
50%	4.000000	8.500000	7.000000	58.000000	2.017080e+07	64000.000000	408.000000
75%	5.000000	10.000000	7.000000	73.000000	2.017123e+07	80000.000000	528.000000
max	10.000000	20.000000	10.000000	118.000000	2.018073e+07	120000.000000	714.000000

Here it is already possible to notice a few discrepancies to give a better look, which will be done in the net section. However, here it is seeable that the max age is 118, which is uncommon, but still feasible; it is valid to take a deeper look into the age distribution. Besides that, it can be seen that few data is missing when income is looked, evaluating those distributions will feed insights about the best way to handle these data.

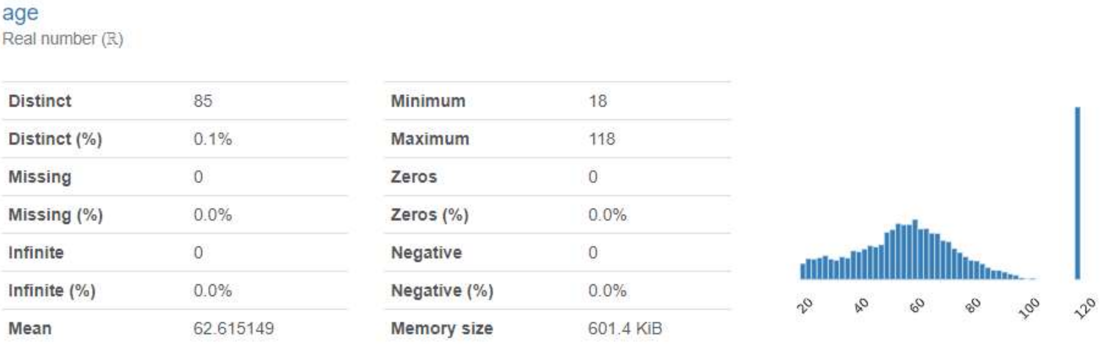
2.2. Exploratory Visualization

For exploring visualization was used graph technics in order to get insights from the data.

- Age

The first variable to be looked at will be age, especially because it was seen that mean age was kinda high and it seemed to have few outliers with age 118.

Figure 7. Age statistics



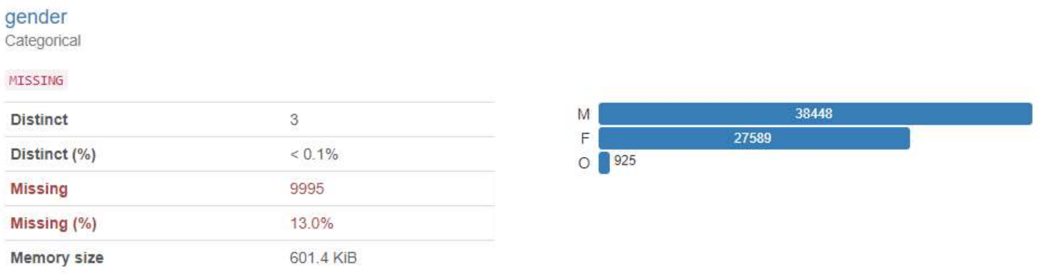
age
Real number (ℝ)

Distinct	85	Minimum	18
Distinct (%)	0.1%	Maximum	118
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	62.615149	Memory size	601.4 KiB

As it is possible to confirm, it seems to have a typo or something with the age 118. Ways of approaching that would be: dropping those registries, segmenting that into a new group, creating a new feature or applying quantiles.

- Gender

Figure 8. Gender statistics



gender
Categorical

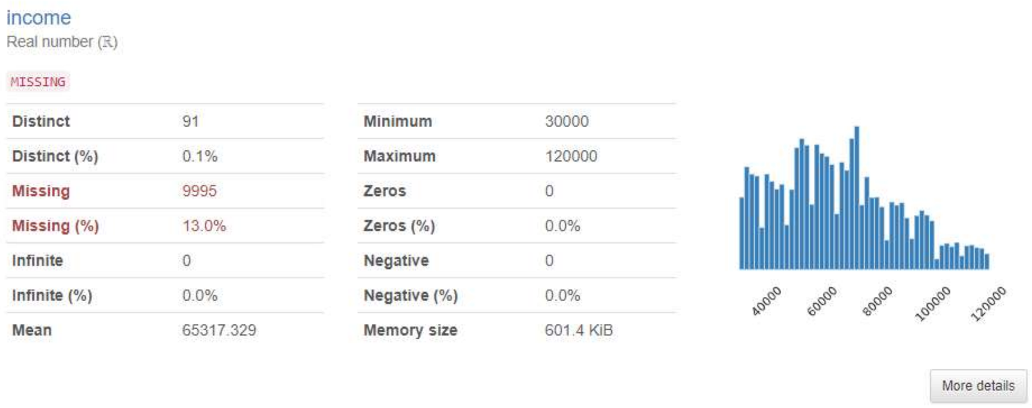
MISSING

Distinct	3
Distinct (%)	< 0.1%
Missing	9995
Missing (%)	13.0%
Memory size	601.4 KiB

It has a group declare as “O”, since gender will be treat as a categorical feature, “O” will be kept and instead of using only F/M, F/M/O will be used to train the model.

- Income

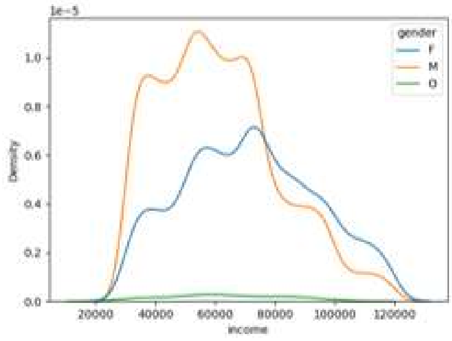
Figure 9. Income statistics



income
Real number (ℝ)

MISSING

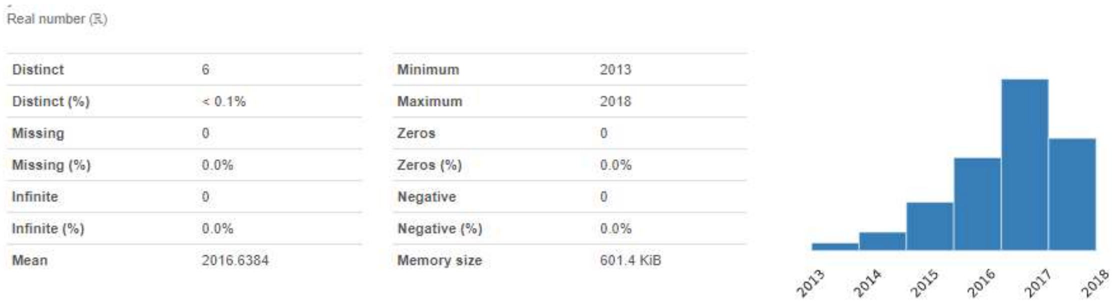
Distinct	91	Minimum	30000
Distinct (%)	0.1%	Maximum	120000
Missing	9995	Zeros	0
Missing (%)	13.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	65317.329	Memory size	601.4 KiB



It was seen before that few data points were missing regarding the income, because of that those data will need to be handle in order to train the model. Besides that, it is seen a segmentation of income between ages, M has a more skew distribution, tending to earn less as W.

- Year

Figure 10. Year statistics



Real number (ℝ)

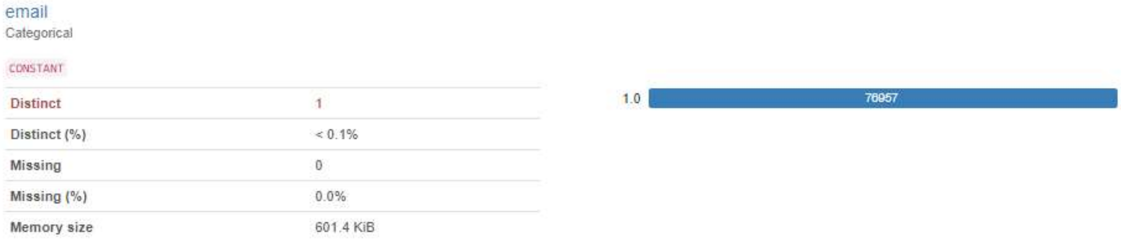
Distinct	6
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	2016.6384

Minimum	2013
Maximum	2018
Zeros	0
Zeros (%)	0.0%
Negative	0
Negative (%)	0.0%
Memory size	601.4 KiB

Year has also a skew distribution; it has a greater concentration on year 2017. If data points were missing, an approach could be replacing it for 2017.

- Email

Figure 11. Email statistics



email
Categorical

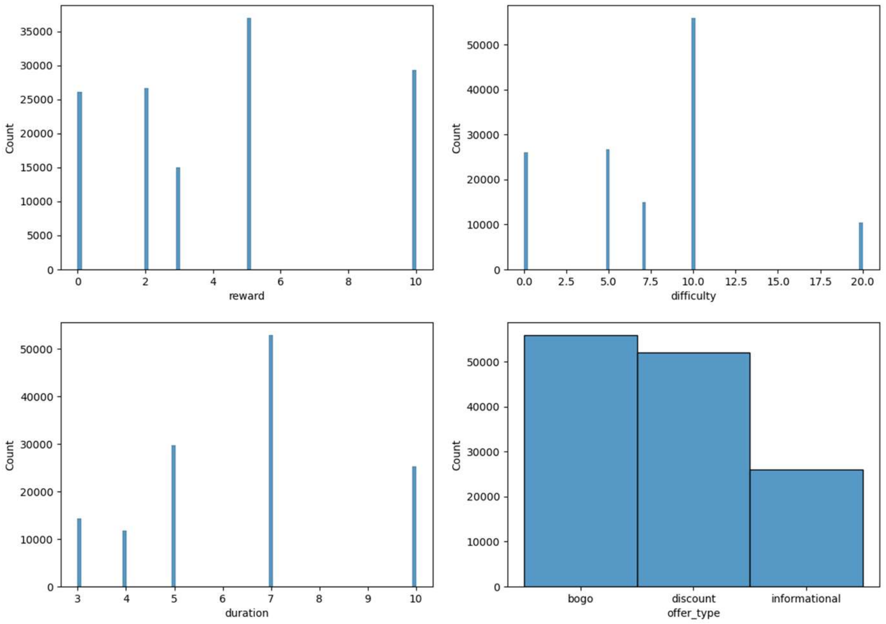
CONSTANT

Distinct	1
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	601.4 KiB

Taking into consideration, completed & uncompleted offers, it is seen that all were available per email, due to this fact this feature does not say anything and could be easily dropped.

- Offer Events

Figure 12. Offer events statistics



It is possible to see among the offers' events that they are not uniform, but are not extremely disproportional. It is important to note that the normal lifecycle of offers is 7 days, this value will be used in processing data to track customer journey.

2.3. Algorithms and Techniques

One key element to mention here is how the case was built. What is seek here to be evaluated is the customer journey, whether the client has completed the offer or not, trying to predict it the person will complete the offer or not before sending it.

It is also needed to be careful to avoid data leakage when crossing the data for preparing the training. One feature that it was used was conversion rate by each customer for each type of offer when trying to predict if the offer would be fulfilled or not. Time needs to be crossed, because when sending the offer it is not known few events.

For example, client x, has a conversion of 80 % for bogo offers, looking promising sending him/her an offer of this type, but that is only known after all the offers were sent, due to that the following table needs to be respected.

Table 1. Conversion rate flow demonstrated

Time	Offer Id	Bogo CR	Has converted	Comments
0	1	50 %	Yes	Set as 50 % as a cold start
100	2	100 %	Yes	Only 1 passed event
200	2	100 %	No	-
500	1	66 %	Yes	-
800	7	75 %	Yes	-

Conversion now is 80 %. It seems obvious that, but it is quite missed in companies, very prone to error and misleading models.

Besides that, all customers offers journeys were tracked through the code built, considering time, like presented in *Figure 1*. For transactions, only its conversion rate was considered.

- ## Data Preprocessing

For data preprocessing a pipeline was built in order to render best data to train, it will be comment in more details in the section 3.1.

Nevertheless, the methods used to handle the data were:

- **Categorical Data:** SimpleImputer + One Hot Encoding (taking into consideration the most frequent value);
- **Continuous Data:** SimpleImputer + StandardScaler (taking into consideration the mean value);
- **Discrete Data:** SimpleImputer + MinMaxScaler (taking into consideration the most frequent value)

It is valid to mention that age could be treat differently, ad hoc tests were done separately but it did not make a significant difference.

- Model

The model that it was used was LightGBM, which is a gradient boosting ensemble method based on decision trees. It can be used for classification and regression (this project is about classification).

LightGBM employs a histogram-based approach, wherein data is grouped into bins using a distribution histogram. Rather than processing each individual data point, the algorithm iterates over these bins, calculating gain and performing data splits. This methodology is adaptable for optimizing sparse datasets. Additionally, LightGBM features exclusive feature bundling, a technique where the algorithm combines exclusive features to diminish dimensionality, resulting in enhanced speed and efficiency.

Due to those facts, it is explained why it did not make a huge difference handling age outliers, that were seen before.

Other models were tested, like neural network, random forest, others, but the one that performed the best and also faster was LightGBM, combining the best of both. Along with the model in the pipeline, has an auto tuning with GridSearch in order to improve performance.

2.4. Benchmark

As a benchmark, business rules were used comparing to it, in order to convince the stakeholders. It is common practices trying to prove to the products owners, that they should apply the model. Instead of the insights that they have about the business, sometimes their insights are actually better than a MVP, because they know so much about the business. With this knowledge and modeling a lot can be done in order to improve the business.

Few simple rules were here applied considering the most important variables, which were outputted by the SHAP evaluation.

- Reward;
- Social;
- Year of membership;
- Income.

What was done here was raising flags whether an offer should be sent or not. If the client had **one flag or more**, the offer is sent. In order to build this business rules a group by was done for each offer type, if the completion rate was above 50 % the flag is raised.

The categorical/boolean variables like reward, social and year of membership had their group by stratified by offer, for income, it was only set the threshold whether the offer should be sent or not.

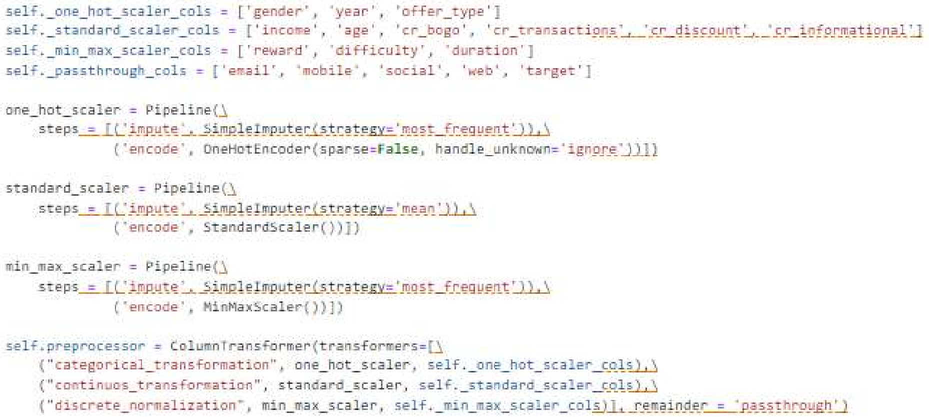
```
{
        'reward': {
                'discount': {2: 0, 3: 1, 5: 0},
                'bogo': {5: 0, 10: 0},
                'informational': {0: 1}
                    },
        'year': {
                'discount': {2013: 1, 2014: 1, 2015: 0, 2016: 1, 2017: 0, 2018: 0},
                'bogo': {2013: 0, 2014: 0, 2015: 0, 2016: 0, 2017: 0, 2018: 0},
                'informational': {2013: 1, 2014: 1, 2015: 1, 2016: 1, 2017: 1, 2018: 1}
                },
       'social': {
                'discount': {0: 0, 1: 1},
                'bogo': {0: 0, 1: 0},
                'informational': {0: 0, 1: 1}
                  },
      'income': 74000.0
}
```

3. Methodology

3.1. Data Preprocessing

Below follows, how each column was handled.

Figure 13. Data processing flow code

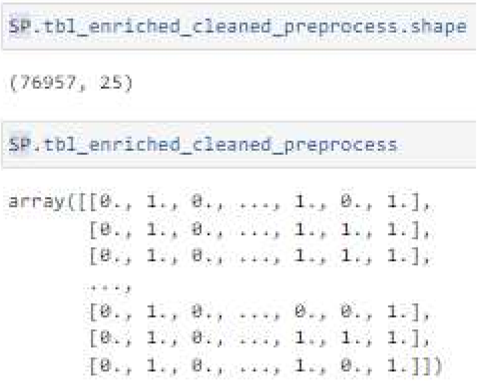


```
self._one_hot_scaler_cols = ['gender', 'year', 'offer_type']
self._standard_scaler_cols = ['income', 'age', 'cr_bogo', 'cr_transactions', 'cr_discount', 'cr_informational']
self._min_max_scaler_cols = ['reward', 'difficulty', 'duration']
self._passthrough_cols = ['email', 'mobile', 'social', 'web', 'target']

one_hot_scaler = Pipeline(\
    steps = [('impute', SimpleImputer(strategy='most_frequent')),\
             ('encode', OneHotEncoder(sparse=False, handle_unknown='ignore'))])

standard_scaler = Pipeline(\
    steps = [('impute', SimpleImputer(strategy='mean')),\
             ('encode', StandardScaler())])

min_max_scaler = Pipeline(\
    steps = [('impute', SimpleImputer(strategy='most_frequent')),\
             ('encode', MinMaxScaler())])

self.preprocessor = ColumnTransformer(transformers=[\
    ("categorical_transformation", one_hot_scaler, self._one_hot_scaler_cols),\
    ("continuos_transformation", standard_scaler, self._standard_scaler_cols),\
    ("discrete_normalization", min_max_scaler, self._min_max_scaler_cols)], remainder = 'passthrough')
```

Even with LightGBM being less sensitive to scaling like a neural network for instance, data preprocessing was kept, the improvement was around 2 % of accuracy performance.

It is remarkable how robust LightGBM is when handling with a restrict amount of data, sides could turn in favor of neural network if more data was available.

Figure 14. Array with data preprocessed



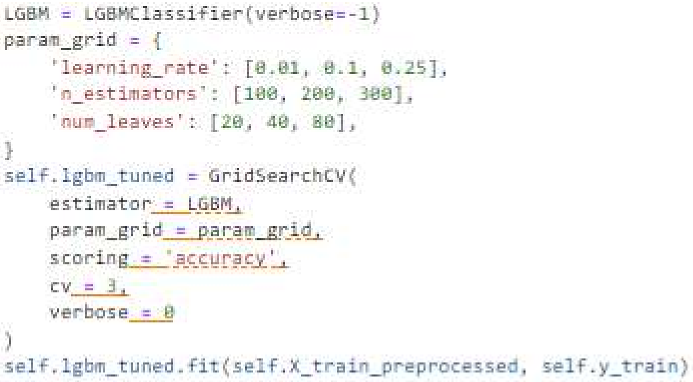
```
SP.tbl_enriched_cleaned_preprocess.shape

(76957, 25)

SP.tbl_enriched_cleaned_preprocess

array([[0., 1., 0., ..., 1., 0., 1.],
       [0., 1., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 1., 1., 1.],
       ...,
       [0., 1., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 1., 0., 1.]])
```

In the image above it is seen that the amount of columns increase due to the usage of one hot encoding.

3.2. Implementation

After the data is preprocessed, the architecture of the model is implemented. A snippet of the code that is responsible for this task follows below.

Figure 15. Pipeline of training and tuning the model



```
LGBM = LGBMClassifier(verbose=-1)
param_grid = {
    'learning_rate': [0.01, 0.1, 0.25],
    'n_estimators': [100, 200, 300],
    'num_leaves': [20, 40, 80],
}
self.lgbm_tuned = GridSearchCV(
    estimator = LGBM,
    param_grid = param_grid,
    scoring = 'accuracy',
    cv = 3,
    verbose = 0
)
self.lgbm_tuned.fit(self.X_train_preprocessed, self.y_train)
```

Like mention before, the model is auto tuned; it consists of pipeline connecting GridSearch and LightGBM on the same pipeline.

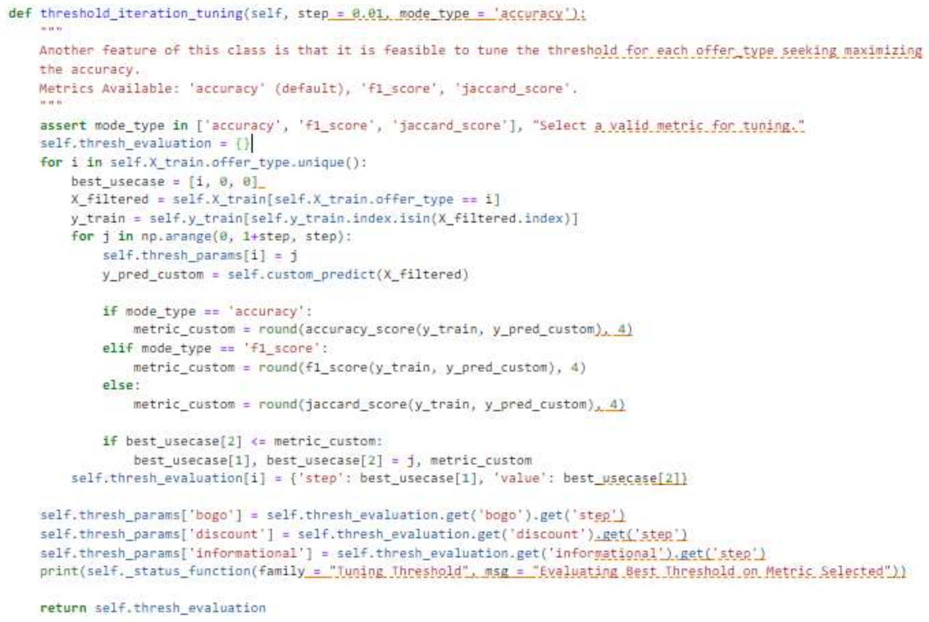
For the tuning was set a small subset of parameters to be tuned: learning rate, number of estimators and a minimum amount per leave. The main goal is to reach the best accuracy that was the metric set on scoring at the GridSearch, later on recall and precision will be looked to per each type of offer, in order to see if it makes sense the output reached.

Any complications has happened until this point. The only thing that is valid to mention is if a larger range of parameters need to be tested, it is recommended to change the type of search; otherwise, it can take a long time until it is finished, or after the first training, the range of parameters is reduced for future tunings. It is important to think about future tunings and automatize that to avoid any data drift that can happen to your data.

3.3. Refinement

One refinement that it was performed to see if the accuracy could be increase, was testing different thresholds per each class, making a custom predictor. A snippet of the code can be looked below.

Figure 16. Function that does the refinement of threshold for each class



```
def threshold_iteration_tuning(self, step = 0.01, mode_type = 'accuracy'):
    """
    Another feature of this class is that it is feasible to tune the threshold for each offer_type seeking maximizing
    the accuracy.
    Metrics Available: 'accuracy' (default), 'f1_score', 'jaccard_score'.
    """
    assert mode_type in ['accuracy', 'f1_score', 'jaccard_score'], "Select a valid metric for tuning."
    self.thresh_evaluation = {}
    for i in self.X_train.offer_type.unique():
        best_usecase = [i, 0, 0]
        X_filtered = self.X_train[self.X_train.offer_type == i]
        y_train = self.y_train[self.y_train.index.isin(X_filtered.index)]
        for j in np.arange(0, 1+step, step):
            self.thresh_params[i] = j
            y_pred_custom = self.custom_predict(X_filtered)

            if mode_type == 'accuracy':
                metric_custom = round(accuracy_score(y_train, y_pred_custom), 4)
            elif mode_type == 'f1_score':
                metric_custom = round(f1_score(y_train, y_pred_custom), 4)
            else:
                metric_custom = round(jaccard_score(y_train, y_pred_custom), 4)

            if best_usecase[2] <= metric_custom:
                best_usecase[1], best_usecase[2] = j, metric_custom
        self.thresh_evaluation[i] = {'step': best_usecase[1], 'value': best_usecase[2]}

    self.thresh_params['bogo'] = self.thresh_evaluation.get('bogo').get('step')
    self.thresh_params['discount'] = self.thresh_evaluation.get('discount').get('step')
    self.thresh_params['informational'] = self.thresh_evaluation.get('informational').get('step')
    print(self._status_function(family = "Tuning Threshold", msg = "Evaluating Best Threshold on Metric Selected"))

    return self.thresh_evaluation
```

It was possible to select different metrics trying to increase performance, establishing different thresholds. When applied, the following results were obtained. However, comparing with the standard LGBM threshold, the performance was kept the same.

```
{
        'informational': {
           'step': 0.53,
           'value': 0.7403},
        'discount': {
           'step': 0.53,
           'value': 0.7527},
        'bogo': {
           'step': 0.54,
           'value': 0.7242}
}
```

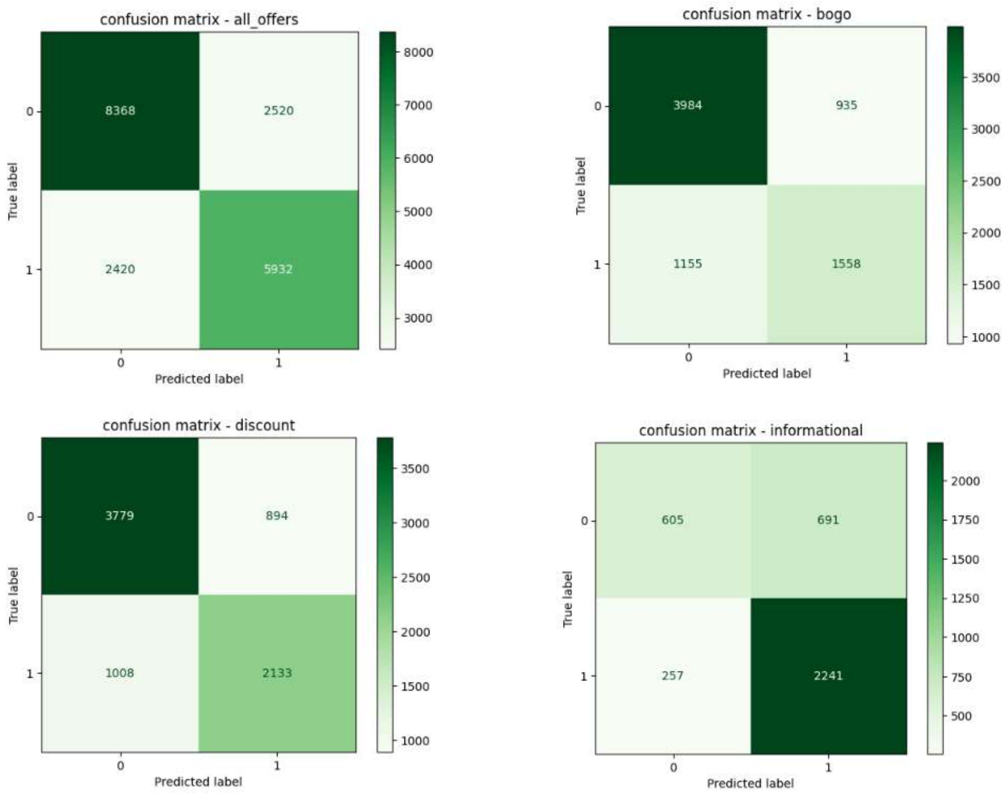
4. Results

4.1. Model Evaluation and Validation

For model evaluation and validation, few metrics were brought, between them it has the confusion matrix, roc-auc curve, precision-racall curve, besides common metrics like accuracy, f1 score, recall, precision and others.

- *Confusion Matrix*

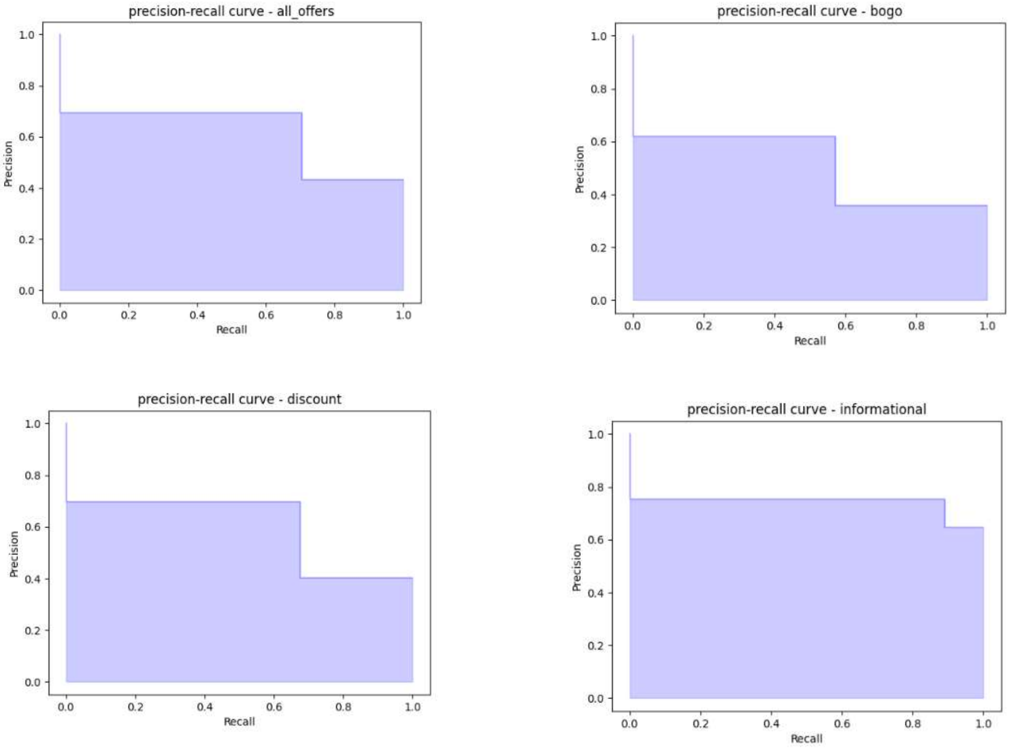
Figure 17. Confusion matrix for each class



It is possible to infer that is consistent the matrix obtained, for the model it is a bit harder to infer bogo and informational correctly than it is for discount. One possible choice to make the customers more fulfilled could be focus for recall with bogo and precision for informational. This way it would be sent less notices, that usually people do not like receiving, and could spam the number of clients using bogo (if that is the purpose, otherwise it is not recommended).

- *Precision-Recall Curve*

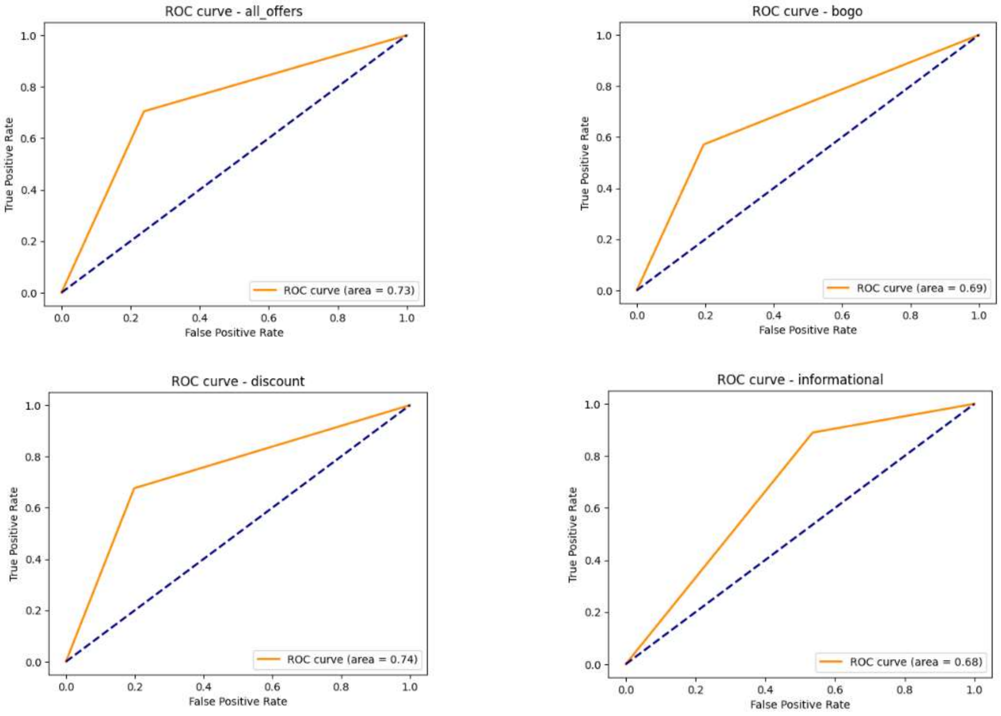
Figure 18. Precision-recall curve for each class



Here it is seeable the trade-off between precision and recall for each type of offer.

- *ROC-AUC Curve*

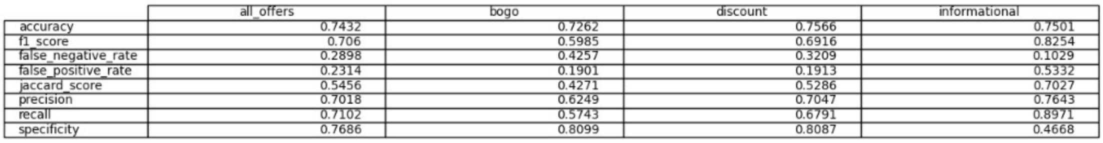
Figure 19. Roc-auc curve for each class



Through those images it can be seen how TPR and FPR behave, the highest point on the curve have similar thresholds as find the dictionary above.

Besides the graphs above, it also has a general metric table.

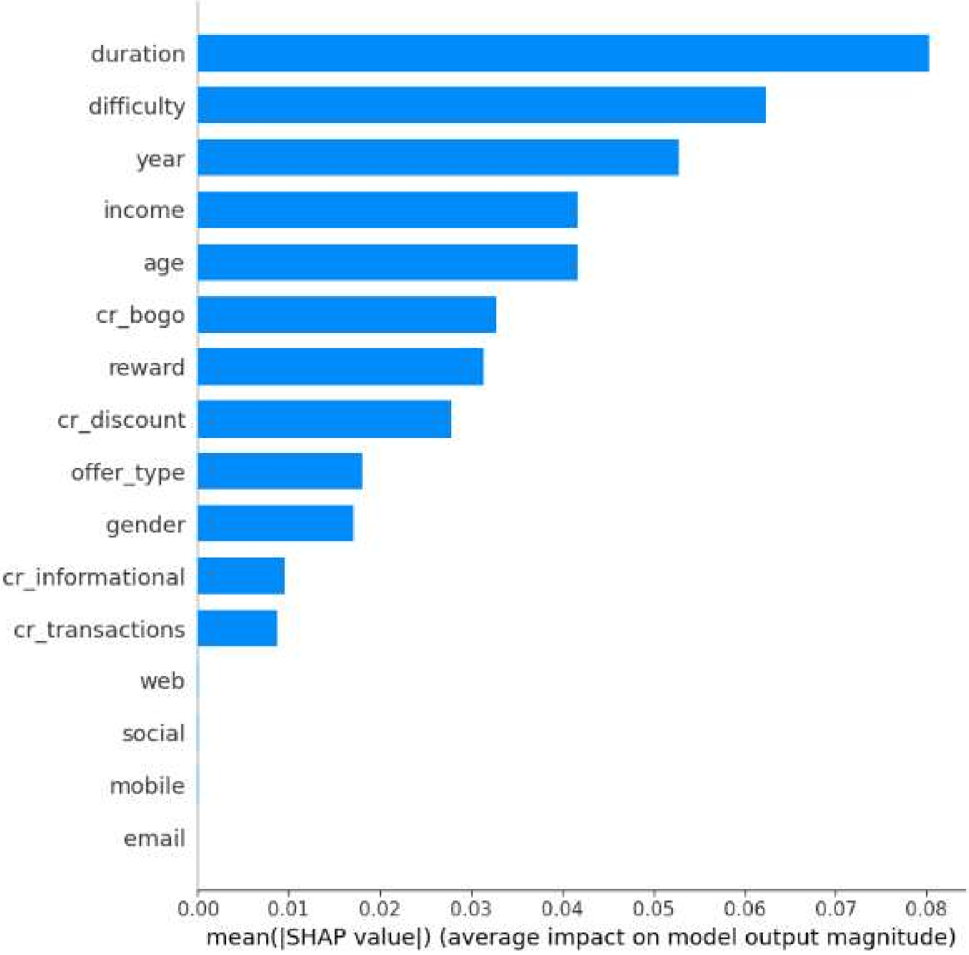
Table 2. General metrics for the model



	all_offers	bogo	discount	informational
accuracy	0.7432	0.7262	0.7566	0.7501
f1_score	0.706	0.5985	0.6916	0.8254
false_negative_rate	0.2898	0.4257	0.3209	0.1029
false_positive_rate	0.2314	0.1901	0.1913	0.5332
jaccard_score	0.5456	0.4271	0.5286	0.7027
precision	0.7018	0.6249	0.7047	0.7643
recall	0.7102	0.5743	0.6791	0.8971
specificity	0.7686	0.8099	0.8087	0.4668

Further than that, it is possible to see the SHAP evaluation to see which of the features are more important.

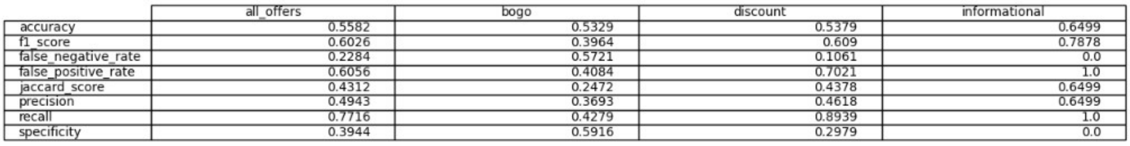
Figure 20. SHAP evaluation for a sample of 200 points



4.2. Justificatiion

As a benchmark for comparison, it was used the rules mention above in section 2.4, below follow the same table as before in order to check the values.

Table 3. General metrics for the business rules



	all_offers	bogo	discount	informational
accuracy	0.5582	0.5329	0.5379	0.6499
f1_score	0.6026	0.3964	0.609	0.7878
false_negative_rate	0.2284	0.5721	0.1061	0.0
false_positive_rate	0.6056	0.4084	0.7021	1.0
jaccard_score	0.4312	0.2472	0.4378	0.6499
precision	0.4943	0.3693	0.4618	0.6499
recall	0.7716	0.4279	0.8939	1.0
specificity	0.3944	0.5916	0.2979	0.0

Taking into consideration as the main metric the accuracy, the model performs better then business rules. This way it is possible to show to the stakeholders this comparison in order for them to buy in the project in order to be used.

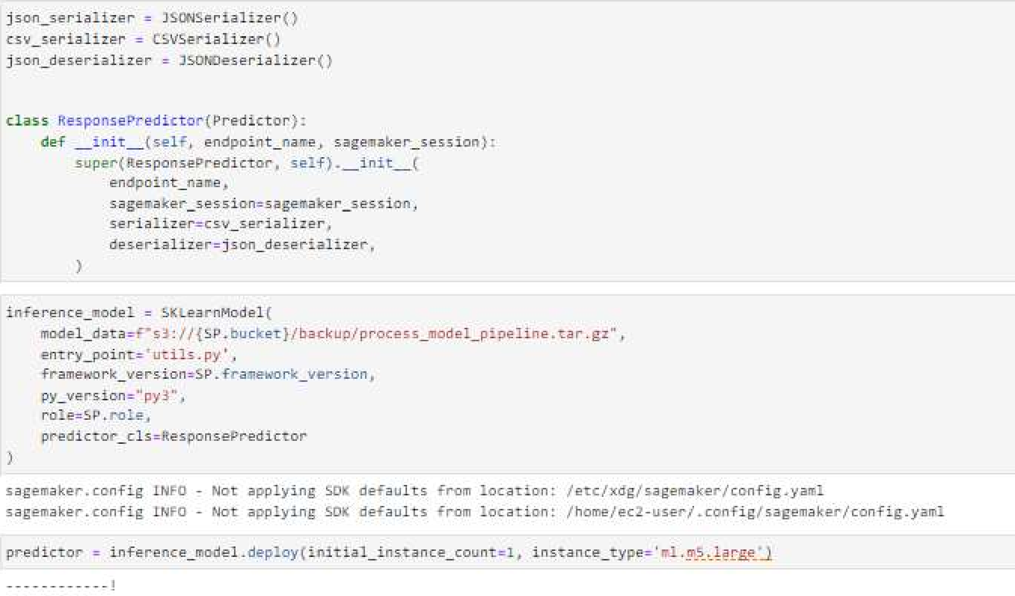
5. Additional

5.1. Deploying Endpoint

In order to the model to be called, it is needed to deploy an endpoint to be called through a lambda function. Due to this fact, the utils.py was built, making it feasible to create an endpoint.

Changes were needed to be done in order to make the endpoint work, some functions were rebuilt. Functions like ColumnsTransformers kept accusing that “feature_names_in_” were missing. Because of that, those functions were rebuilt using BaseEstimator and TransformerMixin. After it, it was possible to invoke the model and through the runtime that was created.

Figure 21. Deploy through sagemaker the endpoint



```
json_serializer = JSONSerializer()
csv_serializer = CSVSerializer()
json_deserializer = JSONDeserializer()


class ResponsePredictor(Predictor):
    def __init__(self, endpoint_name, sagemaker_session):
        super(ResponsePredictor, self).__init__(
            endpoint_name,
            sagemaker_session=sagemaker_session,
            serializer=csv_serializer,
            deserializer=json_deserializer,
        )
```

```
inference_model = SKLearnModel(
    model_data=f"s3://{SP.bucket}/backup/process_model_pipeline.tar.gz",
    entry_point='utils.py',
    framework_version=SP.framework_version,
    py_version="py3",
    role=SP.role,
    predictor_cls=ResponsePredictor
)
```

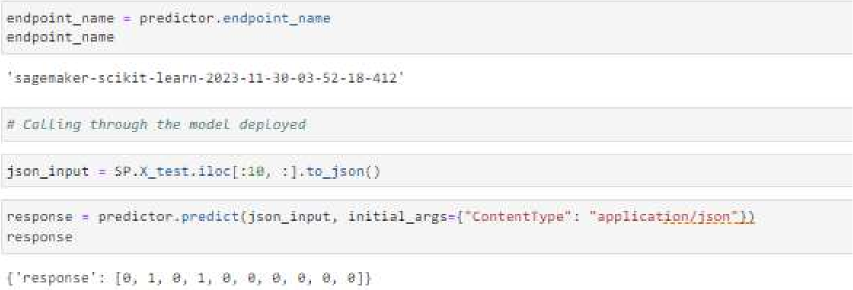
```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

```
predictor = inference_model.deploy(initial_instance_count=1, instance_type='ml.m5.large')
```

```
------------!
```

Few tests were also possible to be done on the notebook.

Figure 22. Testing the use of the endpoint (model)



```
endpoint_name = predictor.endpoint_name
endpoint_name
```

```
'sagemaker-scikit-learn-2023-11-30-03-52-18-412'
```

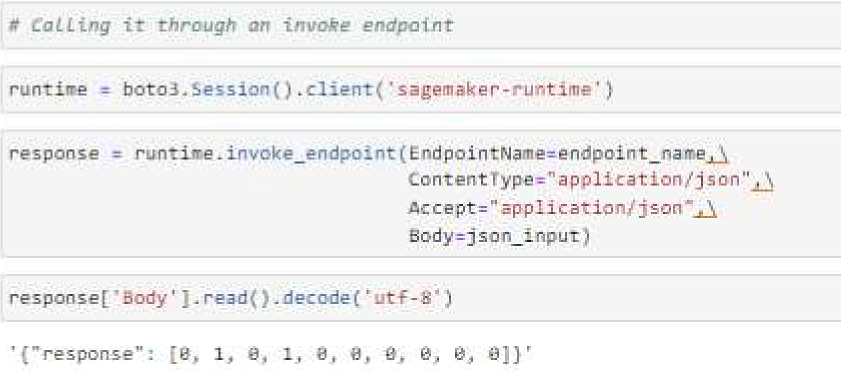
```
# Calling through the model deployed
```

```
json_input = SP.X_test.iloc[:10, :].to_json()
```

```
response = predictor.predict(json_input, initial_args={"ContentType": "application/json"})
response
```

```
{'response': [0, 1, 0, 1, 0, 0, 0, 0, 0, 0]}
```

Figure 23. Testing the use of the endpoint (runtime)



```
# Calling it through an invoke endpoint
```

```
runtime = boto3.Session().client('sagemaker-runtime')
```

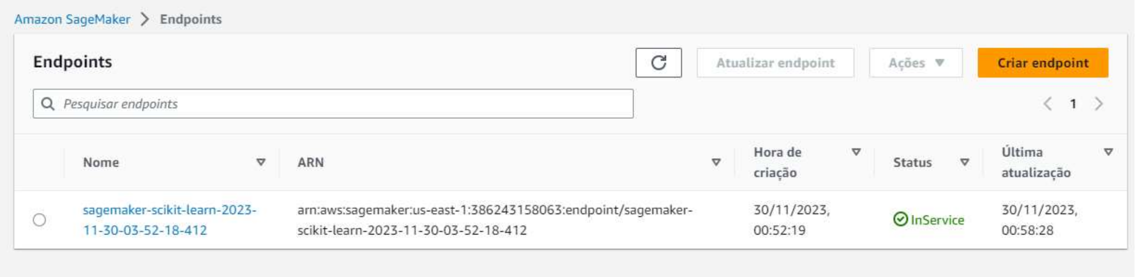
```
response = runtime.invoke_endpoint(EndpointName=endpoint_name,\
                                   ContentType="application/json",\
                                   Accept="application/json",\
                                   Body=json_input)
```

```
response['Body'].read().decode('utf-8')
```

```
'{"response": [0, 1, 0, 1, 0, 0, 0, 0, 0, 0]}'
```

In Amazon SageMaker is also possible to see the endpoint deployed and in service, working normally there.

Figure 24. Endpoint in service

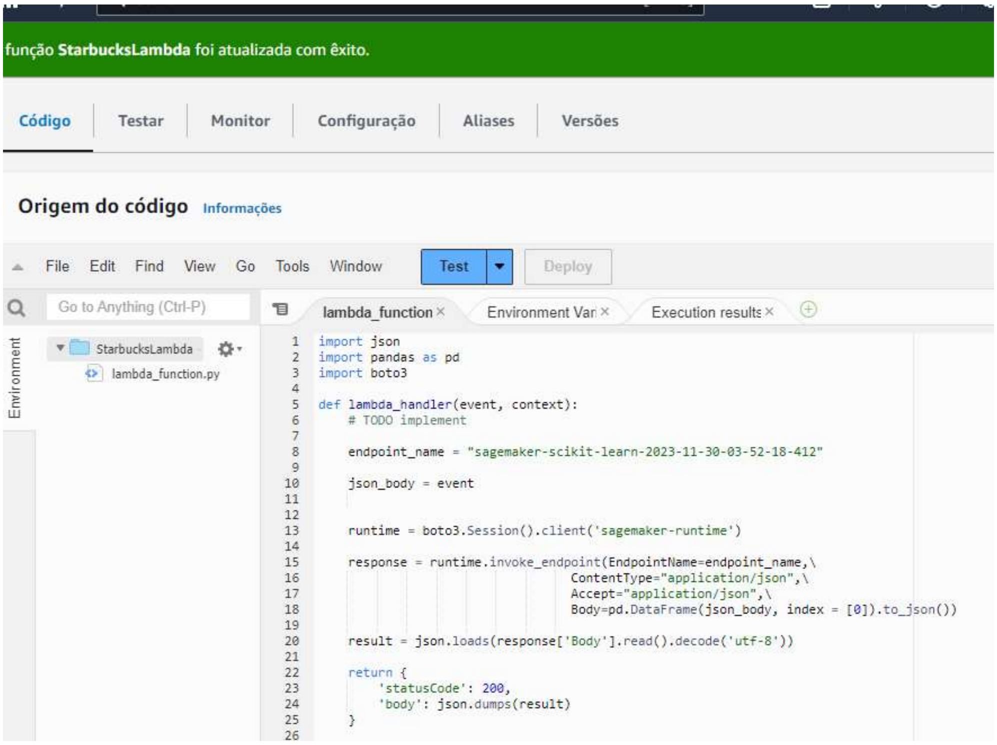


	Nome	ARN	Hora de criação	Status	Última atualização
○	sagemaker-scikit-learn-2023-11-30-03-52-18-412	arn:aws:sagemaker:us-east-1:386243158063:endpoint/sagemaker-scikit-learn-2023-11-30-03-52-18-412	30/11/2023, 00:52:19	✓ InService	30/11/2023, 00:58:28

5.2. Creating Lambda Function

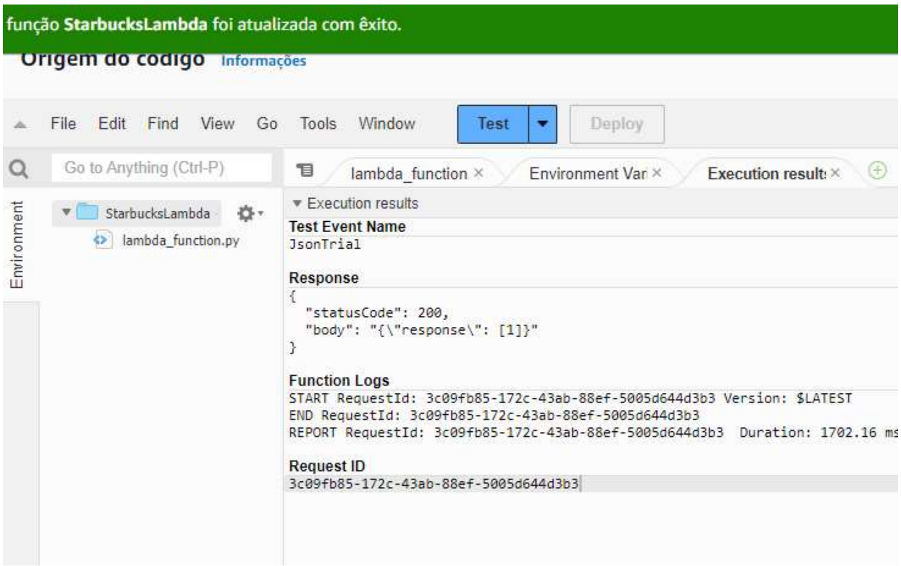
It is valid to mention that in order for using the lambda function, it was needed to change IAM roles, besides that adding a layer that includes pandas in order to use it.

Figure 25. Lambda function



```python
import json
import pandas as pd
import boto3

def lambda_handler(event, context):
    # TODO implement

    endpoint_name = "sagemaker-scikit-learn-2023-11-30-03-52-18-412"

    json_body = event


    runtime = boto3.Session().client('sagemaker-runtime')

    response = runtime.invoke_endpoint(EndpointName=endpoint_name,\
                                       ContentType="application/json",\
                                       Accept="application/json",\
                                       Body=pd.DataFrame(json_body, index = [0]).to_json())

    result = json.loads(response['Body'].read().decode('utf-8'))

    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }

```

Figure 26. Testing lambda function in aws



Test Event Name

```
JsonTrial
```

Response

```
{
  "statusCode": 200,
  "body": "{\"response\": [1]}"
}
```

Function Logs

```
START RequestId: 3c09fb85-172c-43ab-88ef-5005d644d3b3 Version: $LATEST
END RequestId: 3c09fb85-172c-43ab-88ef-5005d644d3b3
REPORT RequestId: 3c09fb85-172c-43ab-88ef-5005d644d3b3  Duration: 1702.16 ms
```

Request ID

```
3c09fb85-172c-43ab-88ef-5005d644d3b3
```