Education Company Analysis Report

Sunny Sun, Cloris Zhang, Yinglu Deng

Motivation

Among the 10,835 customers in the past 10 months, less than 300 customers eventually purchased the service. Yet, the company has 92 full-time sales representatives to collect customers' information and perform routine callbacks to make sales. The extraordinary amount of time spent on data collection and sales call can cost up to ¥10,000,000 every year. With data analysis and machine learning algorithms, we hope to reduce the time and cost by correctly predicting the customers who are likely to purchase the service, which we will refer as a successful customer. After identifying the possible successful customers, we hope to assign them to the right sales representatives. Moreover, we want to provide insights in marketing and product development that can lead to greater sales in the future.

Data

Data Summary

- Timespan: Jan 1st, 2021 - Oct 31th, 2021
- Number of Columns: 190
- Number of Observations: 10835 Customers
- Number of Successful Customers(Signed Contract & Purchased Program): 298
- Total Sales: ¥52,107,050

Data Visualizations & Analysis

Figure 1 illustrates that customers are mainly from coastal areas such as Jiangsu, Shanghai, Zhejiang, and Fujian, and successful clients are mostly in Shanghai, Beijing, and Guangdong.

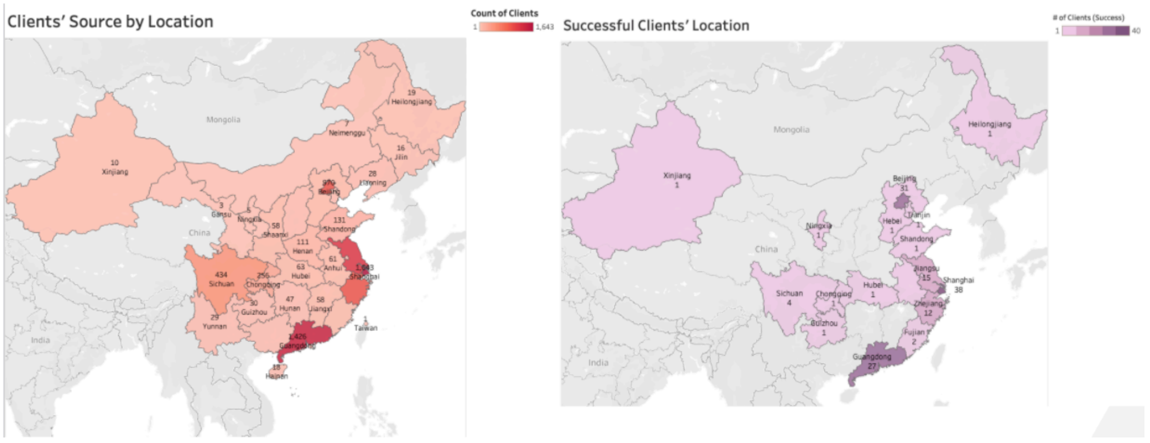


Figure 1: Clients' Geographic Distribution

Despite the diverse geographic locations, most of the customers are collected from media sources like , but most of the customers are collected from media (Figure 2). Hence, we suggest that the company should focus on the successful customer sources and increase its promotion in these areas.

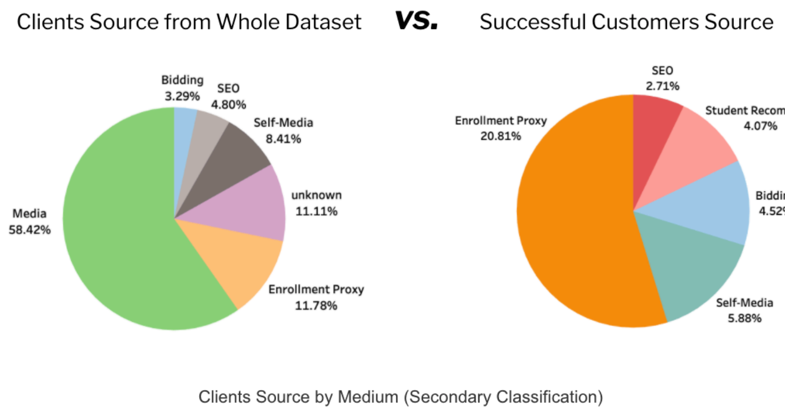


Figure 2: Clients' Source by Medium

Figure 3 demonstrates a huge difference in the proportions of interested programs in customers and successful clients. A majority of customers on file are interested in the DBA program, yet there is little DBA success rate. Most of the successful clients purchased UM series or ISTEC series programs. Hence, we suggest that the company should focus on the development of DBA program as it is a huge demand in the company's existing customers.

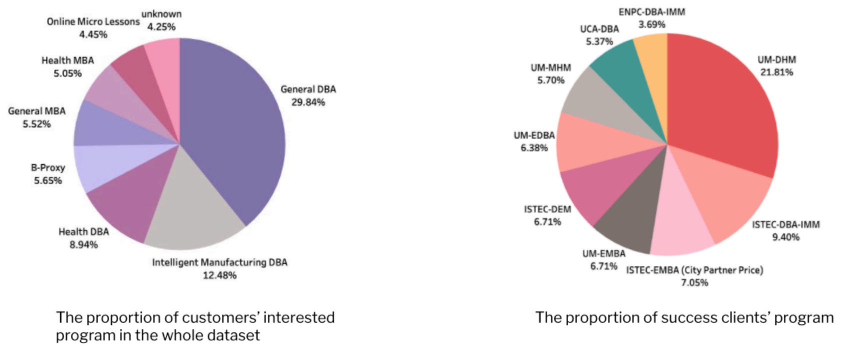


Figure 3: Popularity of Interested Program

Data collection and processing

The data originates from one of the group member's family business in China, which provides many diverse educational programs and services to both businesses and individual customers. The company is currently in the growth and expansion stage, which the company seeks for novel ideas and improvements within the company to facilitate the expansion time.

We selected the customer profile data from this private education company. Due to the limitation in complexity, we decided to focus on the data in a time range of one year. Year of 2021 was selected because it is the newest data and furthermore we can use the model we made to adjust the marketing strategy in the future.

The raw data-set is a combination of ten monthly data-sets in Chinese characters, which was in a different encoding than the default ascii. Hence, we first combined and transformed the data-sets through a series of translations and NLP extractions.After all important information is translated and extracted, we dealt with the dirty data and improved the data quality by changing the inappropriate data types and replacing “unknown” with the missing values.

To make more accurate predictions on the sales’ performance, we extract the actual sales columns to avoid a direct causal relationship with the target variable (success). Also, to avoid over-fitting the data, we need to fix the messy data by deleting columns with similar information and high correlation such as the customer rating and the interest category,which one documents a subjective rating of customer’s success level and the other denotes the frequency of contact according to customer’s interest.

Additionally, for making marketing strategies more efficient and reducing variance, we categorized the names of the specific programs into general groups such as MBA, EMBA, FDBA, etc. To reduce run-time, unexpected variance, the dimensionality of the whole dataset, we also combined columns like most recent DateTime and input DateTime by calculating the delta time.

Analytical Models

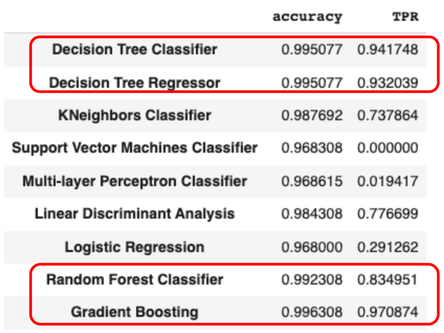
Binary Classification

Our target variable is success (0,1), which is calculated by filtering the clients who have equal or more than one contract signed and/or have a more than ¥0 for total sales amount.

Following are the models with no hyper-parameter tuning that we have tried:

1. Decision Tree Classifier
2. Decision Tree Regressor
3. KNeighbors Classifier
4. Support Vector Machines Classifier
5. Multi-layer Perceptron Classifier
6. Linear Discriminant Analysis
7. Logistic Regression Classifier
8. Random Forest Classifier
9. Gradient Boosting Classifier

In Figure 4, we can see that all of the accuracies are very high, which makes sense because most of the data points are 0, or non-successful customers. Therefore, we believe that accuracy is not a good performance measurement for these models. So we focused on achieving a high true positive rate, correctly identifying the successful customers. Hence, we picked out the outlined models which has a high accuracy and a high TPR to conduct further hyperparameter tuning.

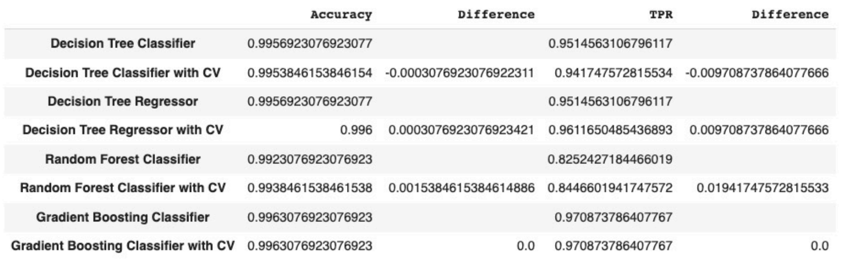


	accuracy	TPR
Decision Tree Classifier	0.995077	0.941748
Decision Tree Regressor	0.995077	0.932039
KNeighbors Classifier	0.987692	0.737864
Support Vector Machines Classifier	0.968308	0.000000
Multi-layer Perceptron Classifier	0.968615	0.019417
Linear Discriminant Analysis	0.984308	0.776699
Logistic Regression	0.968000	0.291262
Random Forest Classifier	0.992308	0.834951
Gradient Boosting	0.996308	0.970874

Figure 4: Base Model Accuracy

Since we want to find the best hyperparameters values to get the perfect prediction results from our model, GridSearchCV is a good way to do so. GridSearch uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. For GridSearchCV, the first argument is the estimator – the four models. The second argument is param_grid – a dictionary with parameter names as keys and lists of parameter values that we want to try. We have experimented with more than five parameters for each model to identify the best fitting model.

Afterward, we use KFold to split the dataset into k consecutive folds with shuffling. Usually, K lies in 5 $\sim$ 10 and we set n_splits to be 5. Then we pass in the number of folds for K-fold cross-validation in GridSearchCV. In the end, we called best_params_ to identify the combination of hyperparameters along with values that gives the best performance of our estimate specified. Random Forest Classifier and Decision Tree Regressor improve their true positive rates by around 2% and 1%. But the true positive rates for Decision Tree Classifier decreases around 1% and Gradient Boosting Classifier stays the same. We hypothesize is that after hyperparameter tuning, some outliers in the successful customers are classifier as non-successful and the already high true positive rates are difficult to maintain and increase (0.95 and 0.97). We are confident in our model as the model can correctly predict the majority of successful customers, and with this model, the sales representatives could save more than 50% of the original time to collect data and identify the customers.



	Accuracy	Difference	TPR	Difference
Decision Tree Classifier	0.9956923076923077		0.9514563106796117	
Decision Tree Classifier with CV	0.9953846153846154	-0.0003076923076922311	0.941747572815534	-0.009708737864077666
Decision Tree Regressor	0.9956923076923077		0.9514563106796117	
Decision Tree Regressor with CV	0.996	0.0003076923076923421	0.9611650485436893	0.009708737864077666
Random Forest Classifier	0.9923076923076923		0.8252427184466019	
Random Forest Classifier with CV	0.9938461538461538	0.0015384615384614886	0.8446601941747572	0.01941747572815533
Gradient Boosting Classifier	0.9963076923076923		0.970873786407767	
Gradient Boosting Classifier with CV	0.9963076923076923	0.0	0.970873786407767	0.0

Figure 5: Advanced Model Accuracy & TPR

Impact & Future Work

- **Seeking potential customers:** By analyzing the successful clients' sources by medium and location, it is easier for the sales associates to find the target customers. They should collect more customers' information in the field like enrollment proxy, self-media, bidding and hold more campaigns in those big cities and coastal cities.

- **Cutting budget and increasing benefit:** The company should decrease the budget like commute fee or time cost by targeting the potential customers. Besides, understanding the needs of the market and industry you are serving is the key to success for operating a business. So, since the majority of clients are interested in the DBA programs, the company needs to improve the quality of these programs and sign more contracts to improve the company's sales.

- **Better marketing strategy:** By comparing the programs from different groups of customers, we discover that the trending programs we sell are not the most attractive programs to most of the potential customers. So, the company has to stop and think about why it is the case and make a better marketing strategy.

- **Further work:** We can try other hyperparameter optimization frameworks like Optuna to automatically tune the hyperparameters and increase the model performance. Also, we can use NLP to extract key information from Remarks Section to improve customer profiles. In addition, conducting feature importance is a good way to find key variables to reduce the data collection process.

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import csv
import regex as re
```

```
df = pd.read_csv("Sales_English(basic clean).csv")
```

```
df = df.rename(columns=str.lower).rename(columns={'most recent activity': 'most rec
df.head()
```

	customer type	sales representative	province	city	most recent activity date	most recent sales representative	custom pc
0	Individual	Weijingjuan	unknown	unknown	2021-09-15	Weijingjuan	Enti Collect
1	Individual	Lishanshan	unknown	unknown	2021-10-03	Lishanshan	En
2	Individual	Hewei	unknown	unknown	2021-04-13	Hewei	En
3	Individual	Lishanshan	unknown	unknown	2021-08-16	Lishanshan	En
4	Individual	Zhouwei	unknown	unknown	2021-09-24	Zhouwei	En

```
#For ongoing customers, Total Contract Amount is greater than 0 and Amount Recievec
#For past customers, Amount Received is greater than 0 and Total Contract Amount is
successful_customers = df[(df['amount received'] > 0) | (df['total contract amount'
successful_customers.head()
```

	customer type	sales representative	province	city	most recent activity date	most recent sales representative	custo p
2	Individual	Hewei	unknown	unknown	2021-04-13	Hewei	E
14	Individual	Gexiaodan	unknown	unknown	2021-03-09	Zhangmai	E
32	Individual	Morong	unknown	unknown	2021-01-04	Morong	Sout
43	Individual	Gexiaodan	unknown	unknown	2021-01-07	Shilinyu	E
78	Individual	Morong	unknown	unknown	2021-02-24	Morong	Sout

```
num_suc_sales = successful_customers.groupby("sales representative").count()['city'
num_suc_sales.head()
```

```
    sales representative
    Gexiaodan     75
    Lihuanhuan    34
    Zhaokeqing    19
    Fengyanyan    15
    Lirui         11
    Name: city, dtype: int64
```

```
tot_assigned_sales = df.groupby(['sales representative']).count()['customer type']
tot_assigned_sales.head()
```

```
sales representative
Administrator    519
Chenchaohui       29
Chendongmei       41
Chenhaotian        8
Chenlimin        218
Name: customer type, dtype: int64
```

```
sales_perf = num_suc_sales.to_frame().join(tot_assigned_sales).rename({'customer ty
sales_perf['succ/assigned'] = sales_perf['# of successful customer']/sales_perf['#
sales_perf.sort_values(by = 'succ/assigned', ascending = False).head(10)
```

	# of successful customer	# of assigned customer	succ/assigned
sales representative			
Shizhanbin	1	1	1.000000
Gexiaodan	75	106	0.707547
Yangtiequan	2	3	0.666667
Liwen	4	19	0.210526
Lihuanhuan	34	170	0.200000
Mali	2	19	0.105263
Chendongmei	4	41	0.097561
Huangwanyun	3	39	0.076923
Tanxiaoxin	4	54	0.074074

```
successful_customers[['amount received','total contract amount','number of signed c
```

```
       amount received  total contract amount  number of signed contract
2              2000.00               370000.0                          1
14             2000.00                 2000.0                          1
32           114654.40                    0.0                          0
43           161942.40                    0.0                          0
78           152000.00                    0.0                          0
...                ...                    ...                        ...
8871           2000.00                    0.0                          0
9129           2000.00                    0.0                          0
9530           2000.00                    0.0                          0
10114          2000.00                    0.0                          0
10265          5967.28                    0.0                          0

298 rows × 3 columns
```

```
df['success'] = (df['amount received'] > 0) | (df['total contract amount'] > 0)
#df.to_csv(r'Sales_cleaning_sucess.csv', index=False)
```

```
df.columns
```

```
Index(['customer type', 'sales representative', 'province', 'city',
       'most recent activity date', 'most recent sales representative',
       'customer pool', 'data creator', 'claim date', 'recent change date',
       'recent editor', 'status', 'belonging department',
       'business registration', 'do-not-disturb', 'number of signed contract',
       'amount received', 'total contract amount', 'account receivable',
       'non-collectibles', 'amount to be collected', 'actual amount collected
       'effective sales amount', 'total sales amount', 'finished customer',
       'customer rating', 'interest category', 'customer source',
       'source secondary classification', 'source third classification',
       'source fourth classification', 'business category', 'interest program
       'information date', 'customer source star rating', 'dududu? (yes/no)',
       'source code', 'customer update status', 'percentage',
       'program sales total', 'expected sales', 'number of calls',
       'repeated? (yes/no)', 'repeated date', 'most recent publish date',
       'success'],
      dtype='object')
```

```
for column in df.select_dtypes(exclude=['int64']):
    print(column, df[column].unique())
```

```
customer type ['Individual' 'Channel' 'Education Abroad' 'Corporate' 'C+B']
sales representative ['Weijingjuan' 'Lishanshan' 'Hewei' 'Zhouwei' 'Lirui' 'L
 'Daiminmin' 'Leiduanyu' 'Zhangmai' 'Gexiaodan' 'Xuyan' 'Tangyiqiong'
 'Wengxiaowen' 'Wanglanmei' 'Maobeilei' 'Xiebingbing' 'Linna - Resign'
 'Yanying' 'Fengyanyan' 'Libin' 'Morong' 'Zhangping' 'Lihuanhuan'
 'Chenchaohui' 'Lidexiang' 'Hulan' 'Xujun' 'Chenlimin' 'Sunningning'
 'Hutianhong' 'Qianshujia' 'Huanglidi' 'Fuguoliang' 'Dujia' 'Wangyifan'
 'Dingzhenhua' 'Xuyanli' 'Gonglijun' 'Xiebin' 'Yurong' 'Zhaokeqing'
 'Jiangjin' 'Zhuyong' 'Yangtangzan' 'Wangjian' 'Mawanli' 'Libo' 'Raoyizhi'
 'Mashuya' 'Zhangyao' 'Shengboyang' 'Fuyu' 'Gujiong' 'Wanglili' 'Wanghong'
 'Liwen' 'Huangwanyun' 'Zhanghao' 'Administrator' 'Donghaosai'
 'Chendongmei' 'Jiajing' 'Jinning' 'Xiaodong' 'Yangtiequan' 'Honghaixia'
 'Shizhanbin' 'Zhouhualin' 'Wumingzhu' 'Yangzhan' 'Luosiyi' 'Chenhaotian'
 'Zhouhonglin' 'Mali' 'Wucheng' 'Youmiao' 'Quwenpeng' 'Linlin' 'Linxieguo'
 'Tanxiaoxin' 'Pengxiaogui' 'Jiangyongyong' 'Jiangxiao' 'Sunliping'
 'Songhong' 'Zhaoweizheng' 'Zhangben' 'Xujingxian' 'Chenyuzhen' 'Dengqing'
 'Tangzijun']
province ['unknown' 'Shanghai' 'Guangdong' 'Sichuan' 'Xinjiang' 'Beijing' 'Ji
 'Shandong' 'Jilin' 'Zhejiang' 'Henan' 'Fujian' 'Liaoning' 'Shanxi'
 'Jiangxi' 'Hubei' 'Hebei' 'Guangxi' 'Chongqing' 'Shaanxi' 'Anhui' 'Hunan'
 'Guizhou' 'Yunnan' 'Hainan' 'Tianjin' 'Neimenggu' 'Hongkong'
 'Heilongjiang' 'Ningxia' 'Gansu' 'Taiwan']
city ['unknown' 'Shanghai' 'Beijing' 'Zhengzhou' 'Suzhou' 'Xiamen' 'Taizhou'
 'Nantong' 'Zibo' 'Luoyang' 'Zhoukou' 'Jinhua' 'Weifang' 'Jingzhou'
 'Zhuma' 'Shijiazhuang' 'Xianning' 'Shangqiu' 'Changzhou' 'Zhongshan'
```

```
  'Liuzhou' 'Chongqing' 'Huaian' 'Suqian' "Xi'an" 'Yuncheng' 'Nanjing'
  'Shenzhen' 'Qingdao' 'Guangzhou' 'Baoding' 'Tangshan' 'Yantai'
  'Zhanjiang' 'Fuyang' 'Dalian' 'Guilin' 'Langfang' 'Jiaozuo' 'Quanzhou'
  'Jinan' 'Fuoshan' 'Nanning' 'Chongzuo' 'Taiyuan' 'Xinyang' 'Chenzhou'
  'Wuhan' 'Linyi' 'Shaoxing' 'Weinan' 'Hangzhou' 'Wenzhou' 'Zhenjiang'
  'Changsha' 'Dongying' 'Loudi' 'Ganzhou' 'Zaozhuang' 'Nanchang' 'Kunming'
  'Chengdu' 'Hefei' 'Fuzhou' "Tai'an" 'Xuzhou' 'Zunyi' 'Anyang' 'Jining'
  'Siangyang' 'Lvliang' 'Hanzhong' 'Dezhou' 'Jiangmen' 'Nanyang' 'Wuxi'
  'Zhaotong' 'Pingdingshan' 'Shanwei' 'Haikou' 'Bijie' 'Handan' 'Yancheng'
  'Shantou' 'Dongguan' 'Cangzhou' 'Tianjin' 'Yulin' 'Heyuan' 'Xinxiang'
  'Kaifeng' 'Weihai' 'Huainan' 'Huhehaote' 'Linfen' 'Chengde' 'Ezhou'
  'Shenyang' 'Sanya' 'Deyang' 'Ningbo' 'Puti' 'Guangan' 'Baoji' 'Bengbu'
  'Chaozhou' 'Maoming' 'Guiyang' 'Zhaoqing' 'Hongkong' 'Xiaogan'
  'Lianyungang' 'Luohe' 'Aletai' 'Jieyang' 'Meizhou' 'Jiaxing'
  'Qinhuangdao' 'Jinzhong' 'Zhoushan' 'Wuhu' 'Longyan' 'Xingtai' 'Changde'
  'Yibin' 'Yangjiang' 'Heze' 'Datong' 'Rizhao' 'Anqing' 'Huzhou' 'Xianyang'
  'Jincheng' 'Hengyang' 'Xuancheng' 'Xiangtan' 'Changzhi' 'Yangquan'
  'Shiyan' 'Jinzhou' 'Sanming' 'Lishui' 'Hechi' 'Liupanshui' 'Fuxin'
  'Binzhou' 'Zhuhai' 'Tonglin' 'Changchun' 'Huizhou' 'Nanchong' 'Qinzhou'
  'Huangshi' 'Jingdezhen' 'Baoshan' 'Mianyang' 'Anshan' 'Huanggang'
  'Chuxiongyizu' 'Haerbin' 'Yingtan' 'Yuxi' 'Nanping' 'Pingxiang'
  'Shangrao' 'Zhuzhou' 'Jiujiang' 'Zhangjiajie' 'Huangshan' 'Huaihua'
  'Yueyang' 'Quzhou' 'Leshan' 'Liaocheng' 'Qianxinanbu' 'Shaoyang'
  'Yangzhou' 'Jiamusi' 'Yinchuan' 'Huaibei' 'Ganzi' 'Zigong' 'Ziyang'
  'Daqing' 'Suining' 'Mudanjiang' 'Tianshui' "Bayannao'er" 'Dazhou' 'Habi'
  'Fushun' "Ya'an" 'Jingmen' 'Wulumuqi' 'Jilin' 'Baotou' 'Liuan' 'Luzhou'
  'Xinyu' 'Wuzhong' 'Liaoyang' 'Yili' 'Putian' 'Xuchang' 'Chuzhou'
  'Hengshui' 'Kashi' 'Shaoguan' 'Taiwan' 'Meishan' 'Puyang' 'Bazhong'
  'Baise' 'Heihe' 'Panzhihua' 'Lanzhou' 'Yunfu' "Pu'er" 'Huludao'
  'Maanshan']
most recent activity date ['2021-09-15' '2021-10-03' '2021-04-13' '2021-08-16
  '2021-05-28' '2021-09-03' '2021-10-09' '2021-09-26' '2021-09-18'
  '2021-01-06' '2021-01-04' '2021-01-29' '2021-03-09' '2021-06-02'
  '2021-06-29' '2021-08-30' '2021-10-08' '2021-01-07' '2021-03-18'
  '2021-01-26' '2021-04-02' '2021-09-28' 'unknown' '2021-03-12'
```

```
#find numerical columns & categorical columns
sales_cols = [column for column in df.select_dtypes(include='number')][:-2]
print("Sales columns", sales_cols)
```

```
Sales columns ['number of signed contract', 'amount received', 'total contract
```

```
#Find datetime columns in data
date_cols = df.rename(columns=str.lower).filter(regex=(".*(date)+.*")).drop(['custc
print("Datetime columns", date_cols)
```

```
Datetime columns Index(['most recent activity date', 'claim date', 'recent chā
       'information date', 'repeated date', 'most recent publish date'],
      dtype='object')
```

```
date_sales_cols = list(sales_cols)+list(date_cols)
cat_cols = [column for column in df.drop(columns = date_sales_cols ,axis=1)]
cat_cols
```

```
['customer type',
 'sales representative',
 'province',
 'city',
 'most recent sales representative',
 'customer pool',
 'data creator',
 'recent editor',
 'status',
 'belonging department',
 'business registration',
 'do-not-disturb',
 'customer rating',
 'interest category',
 'customer source',
 'source secondary classification',
 'source third classification',
 'source fourth classification',
 'business category',
 'interest program',
 'customer source star rating',
 'dududu? (yes/no)',
 'source code',
 'customer update status',
 'percentage',
 'program sales total',
 'expected sales',
 'number of calls',
 'repeated? (yes/no)',
 'success']
```

```
#Change program sales to numeric
df['program sales total'] = df['program sales total'].map(lambda x: x.lstrip('\t'))
df['program sales total'].head()
```

```
    0    588000
    1    588000
    2    268000
    3    588000
    4    300000
    Name: program sales total, dtype: int64
```

```
df['interest category'].unique()
```

```
    array(['D', 'F', 'B', 'C', 'G', 'Official Student', 'W', 'A', 'I', 'E',
           'X', 'J', 'unknown', 'H', 'R'], dtype=object)
```

```
#F-paid interview fee, A - 1/week, B - 1/2 weeks, C - 1/3 weeks, D - 1/2 months
#E - EMC acitivity interest/No particular interest, #G - 未接, X - 错号, H - 空号,
#Official Student, #W - 有兴趣没对应项目
#Official Student - 100, F - 80, A - 60, B - 50, C - 40, D -25, W - 15, E,G - 10, X
num_int = df['interest category'].replace({'J':'I','R':'I','Official Student':100,
                               'W':15, 'E':10, 'G':10, 'W':10, 'I':10, 'X':0, 'H':
```

```
r = np.corrcoef(df['customer rating'], num_int)
r
#Highly correlated, drop interest category
cat_cols.remove('interest category')
```

```
cat_cols = [cols for cols in cat_cols if cols != 'interest category' and cols != 'p
cat_cols
```

```
['customer type',
 'sales representative',
 'province',
 'city',
 'most recent sales representative',
 'customer pool',
 'data creator',
 'recent editor',
 'status',
 'belonging department',
 'business registration',
 'do-not-disturb',
 'customer rating',
 'customer source',
 'source secondary classification',
 'source third classification',
 'source fourth classification',
 'business category',
 'interest program',
 'customer source star rating',
 'dududu? (yes/no)',
 'source code',
 'customer update status',
 'program sales total',
 'expected sales',
 'number of calls',
 'repeated? (yes/no)']
```

```
df['number of calls'] = df['number of calls'].replace('unknown',0).astype(int)
df['dududu? (yes/no)'] = df['dududu? (yes/no)'].astype('int')
df['repeated? (yes/no)'] = df['repeated? (yes/no)'].replace({'Yes':1,'unknown':0})
```

```
X = df[cat_cols]
y = df['success']

cnt_int_prog = pd.DataFrame(X.groupby(['interest program']).count().sort_values(by
cnt_int_prog.index
```

```
Index(['General DBA', 'Intelligent Manufacturing DBA', 'Health DBA', 'B-Proxy
       'General MBA', 'Health MBA', 'Online Micro Lessons', 'unknown',
       'Education DBA', 'Finance DBA', 'UM-EMBA', 'Data Tranformation DBA',
       'Data Econ DBA', 'UM-DHM', 'Study Abroad Phd', 'Others', 'UM-EDBA',
       'ISTEC-EMBA (City Partner Price)', 'Asset Management DBA', 'ISTEC-DBA',
       'UCA-DBA', 'ISTEC-DBA-IMM', 'ISTEC-MBA', 'ISTEC-DEM', 'UM-MHM',
       'Intelligent Manufacturing MBA', 'ENPC-DBA-IMM', 'Data Science DBA',
       'Montepellier - EMBA', 'NEOMA-DBA', 'Study Abroad - Bus',
       'ISTEC-MBA-IMM', 'NEOMA-DDE', 'DIU-MBA HCM', 'ISTEC-MBA-OL',
       'ISTEC-DDT', 'ISTEC-FDBA', 'NEOMA-DAM', 'ENPC-DIMM', 'Culture DBA',
       'ENPC-DHM', 'Study Abroad Master', 'Wealth Management DBA',
       'ISTEC-DBA-OL', 'B-Partner', 'Data Marketing DBA', 'Online MBA',
       'UM-PHD(Study Abroad, Require pre-PHD)', 'UM-PHD', 'ESC-BBA-OL',
       'Online DBA', 'NEOMA-DDM', 'NEOMA-FDBA', 'Finance MBA', 'RFA',
       'IUKL-PBA (Business Management)', 'CGFT', 'IS-FMBA', 'ISTEC-DCM',
       'ISTEC-EMBA (Old Price)', 'UCA-DBA-HK', 'CL-MIB', 'Nice DBA - Zhongxu',
       'UCA-EMBA', 'Business MBA', 'UM-(Master & Phd)', 'PHD (Study Abroad)',
       'BIFF', 'ENPC-DBA', 'NEOMA-PHD', 'ISTEC-FMBA',
       'UCA-EMBA (City Partner Price)', 'UCA-DBA-SZ', 'Study Abroad Bachelor',
       'IUKL-PED （Education）', 'NEOMA-DBA-SZ', 'Online-B', 'NEOMA-DWM',
       'Nice MBA', 'IUKL-PCO （Communication）'],
      dtype='object', name='interest program')
```

```
#generalize program by degree
def generalizeprogram(column):
    for i in range(len(column)):
        if 'FDBA' in column[i]:
            column[i] = 'FDBA'
        elif 'DBA' in column[i]:
            column[i] = 'DBA'
        elif 'FMBA' in column[i]:
            column[i] = 'FMBA'
        elif 'EMBA' in column[i]:
            column[i] = 'EMBA'
        elif 'MBA' in column[i]:
            column[i] = 'MBA'
        elif 'Online' in column[i]:
            column[i] = 'Online'
        elif 'IUKL' in column[i]:
            column[i] = 'IUKL'
        elif 'Study Abroad' in column[i]:
            column[i] = 'Study Abroad'
        elif 'B-' in column[i]:
            column[i] = 'B'
        elif 'NEOMA' in column[i]:
            column[i] = 'NEOMA'
        elif 'unknown' in column[i]:
            column[i] = 'unknown'
        else:
            column[i] = 'Others'

X['interest program'] = generalizeprogram(list(X['interest program']))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:29: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
```

```
# Train Test split
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics

X_dum = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X_dum, y, test_size=0.3, random
```

Model Selection

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)
print("Train accuracy: ", model.score(X_train,y_train))
print("Test accuracy: ", model.score(X_test,y_test))
print(classification_report(y_test,pred))
```

```
Train accuracy:  0.9705921139390743
Test accuracy:  0.968
              precision    recall  f1-score   support

       False       0.98      0.99      0.98      3147
        True       0.49      0.29      0.37       103

    accuracy                           0.97      3250
   macro avg       0.73      0.64      0.67      3250
weighted avg       0.96      0.97      0.96      3250
```

```
# The Decision tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifer object
dtc = DecisionTreeClassifier()
# Train Decision Tree Classifer
dtc.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = dtc.predict(X_test)
# model Evaluation
acc_dtc = accuracy_score(y_test, y_pred)
acc_dtc
```

```
    0.9950769230769231
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

           False       1.00      1.00      1.00      3147
            True       0.91      0.94      0.92       103

        accuracy                           1.00      3250
       macro avg       0.95      0.97      0.96      3250
    weighted avg       1.00      1.00      1.00      3250
```

```
from sklearn.metrics import recall_score
```

```
tpr_dtc = recall_score(y_test, y_pred)
tpr_dtc
```

```
    0.941747572815534
```

```
from sklearn.tree import DecisionTreeRegressor
# build model
dtr = DecisionTreeRegressor()
# fit classifiers
dtr.fit(X_train, y_train)
# Prediction
y_pred = dtr.predict(X_test)

# model Evaluation
acc_dtr = accuracy_score(y_test, y_pred)
acc_dtr
```

```
    0.9950769230769231
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

           False       1.00      1.00      1.00      3147
            True       0.91      0.93      0.92       103

        accuracy                           1.00      3250
       macro avg       0.96      0.96      0.96      3250
    weighted avg       1.00      1.00      1.00      3250
```

```
tpr_dtr = recall_score(y_test, y_pred)
tpr_dtr
```

```
    0.9320388349514563
```

```
# The KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
# build model
knn_model = KNeighborsClassifier()
# fit classifiers
knn_model.fit(X_train, y_train)
# Prediction
y_pred = knn_model.predict(X_test)

# model Evaluation
acc_knn = accuracy_score(y_test, y_pred)
acc_knn
```

```
    0.9876923076923076
```

```
tpr_knn = recall_score(y_test, y_pred)
tpr_knn
```

```
    0.7378640776699029
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

           False       0.99      1.00      0.99      3147
            True       0.85      0.74      0.79       103

        accuracy                           0.99      3250
       macro avg       0.92      0.87      0.89      3250
    weighted avg       0.99      0.99      0.99      3250
```

```
# the SVM Classifier
from sklearn import svm
# build model
svm_model = svm.SVC()
# fit classifiers
svm_model.fit(X_train, y_train)
# Prediction
y_pred = svm_model.predict(X_test)
# model Evaluation
acc_svm = accuracy_score(y_test, y_pred)
acc_svm
```

```
    0.9683076923076923
```

```
tpr_svm = recall_score(y_test, y_pred)
tpr_svm
```

```
    0.0
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

           False       0.97      1.00      0.98      3147
            True       0.00      0.00      0.00       103

        accuracy                           0.97      3250
       macro avg       0.48      0.50      0.49      3250
    weighted avg       0.94      0.97      0.95      3250

    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:130
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:130
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:130
      _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.neural_network import MLPClassifier
mlp_model = MLPClassifier()
# fit classifiers
mlp_model.fit(X_train, y_train)
# Prediction
y_pred = mlp_model.predict(X_test)
# model Evaluation
acc_mlp = accuracy_score(y_test, y_pred)
acc_mlp
```

```
    0.9686153846153847
```

```
tpr_mlp = recall_score(y_test, y_pred)
tpr_mlp
```

```
    0.019417475728155338
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

           False       0.97      1.00      0.98      3147
            True       0.67      0.02      0.04       103

        accuracy                           0.97      3250
       macro avg       0.82      0.51      0.51      3250
    weighted avg       0.96      0.97      0.95      3250
```

```
# Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# build model
lda = LinearDiscriminantAnalysis()
# fit classifiers
lda.fit(X_train, y_train)
# Prediction
y_pred = lda.predict(X_test)
# model Evaluation
acc_lda = accuracy_score(y_test, y_pred)
acc_lda
```

```
    0.9843076923076923
```

```
tpr_lda = recall_score(y_test, y_pred)
tpr_lda
```

```
    0.7766990291262136
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

           False       0.99      0.99      0.99      3147
            True       0.74      0.78      0.76       103

        accuracy                           0.98      3250
       macro avg       0.87      0.88      0.88      3250
    weighted avg       0.98      0.98      0.98      3250
```

```
# The Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression
# build model
log_model = LogisticRegression()
# fit classifiers
log_model.fit(X_train, y_train)
# Prediction
y_pred = log_model.predict(X_test)

# model Evaluation
acc_log = accuracy_score(y_test, y_pred)
acc_log
```

```
    0.968
```

```
tpr_log = recall_score(y_test, y_pred)
tpr_log
```

```
    0.2912621359223301
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       False       0.98      0.99      0.98      3147
        True       0.49      0.29      0.37       103

    accuracy                           0.97      3250
   macro avg       0.73      0.64      0.67      3250
weighted avg       0.96      0.97      0.96      3250
```

```
# The Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
# build model
rf_model = RandomForestClassifier()
# Fitting the classifier
rf_model.fit(X_train, y_train)
# Prediction
y_pred = rf_model.predict(X_test)
# model Evaluation
acc_rf = accuracy_score(y_test, y_pred)
acc_rf
```

```
0.9923076923076923
```

```
tpr_rf = recall_score(y_test, y_pred)
tpr_rf
```

```
0.8349514563106796
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       False       0.99      1.00      1.00      3147
        True       0.91      0.83      0.87       103

    accuracy                           0.99      3250
   macro avg       0.95      0.92      0.93      3250
weighted avg       0.99      0.99      0.99      3250
```

```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
# build model
gbc = GradientBoostingClassifier()
# Fitting the classifier
gbc.fit(X_train, y_train)
# Prediction
y_pred = gbc.predict(X_test)
# model Evaluation
acc_gbc = accuracy_score(y_test, y_pred)
acc_gbc
```

```
    0.9963076923076923
```

```
tpr_gbc = recall_score(y_test, y_pred)
tpr_gbc
```

```
    0.970873786407767
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

           False       1.00      1.00      1.00      3147
            True       0.92      0.97      0.94       103

        accuracy                           1.00      3250
       macro avg       0.96      0.98      0.97      3250
    weighted avg       1.00      1.00      1.00      3250
```

```
df = pd.DataFrame(np.array([[acc_dtc, tpr_dtc], [acc_dtr, tpr_dtr], [acc_knn, tpr_k
                            [acc_log, tpr_log], [acc_rf, tpr_rf], [acc_gbc, tpr_gbc
                  columns=['accuracy', 'TPR'],
                  index=['Decision Tree Classifier', 'Decision Tree Regressor', 'KN
                  'Linear Discriminant Analysis', 'Logistic Regression', 'Random Fo
df
```

	accuracy	TPR
Decision Tree Classifier	0.995077	0.941748
Decision Tree Regressor	0.995077	0.932039
KNeighbors Classifier	0.987692	0.737864
Support Vector Machines Classifier	0.968308	0.000000
Multi-layer Perceptron Classifier	0.968615	0.019417
Linear Discriminant Analysis	0.984308	0.776699
Logistic Regression	0.968000	0.291262
Random Forest Classifier	0.992308	0.834951
Gradient Boosting	0.996308	0.970874

```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

```
grid_values = {'n_estimators': [100], 'learning_rate': [0.1]}
gbc = GradientBoostingClassifier()
gbc_cv = GridSearchCV(gbc, param_grid=grid_values, cv=10, n_jobs=-1)
gbc_cv.fit(X_train, y_train)
```

```
    GridSearchCV(cv=10, estimator=GradientBoostingClassifier(), n_jobs=-1,
                 param_grid={'learning_rate': [0.1], 'n_estimators': [100]})
```

```
y_pred = gbc_cv.predict(X_test)
acc_gbc_cv = accuracy_score(y_test, y_pred)
acc_gbc_cv
```

```
    0.9963076923076923
```

```
y_pred = gbc_cv.predict(X_test)
acc_gbc_cv10 = accuracy_score(y_test, y_pred)
acc_gbc_cv10
```

```
0.9963076923076923
```

```
tpr_gbc_cv = recall_score(y_test, y_pred)
tpr_gbc_cv
```

```
0.970873786407767
```

```
df = pd.DataFrame(np.array([[acc_gbc, tpr_gbc], [acc_gbc_cv, tpr_gbc_cv]]),
                  columns=['accuracy', 'TPR'],
                  index=['Gradient Boosting Classifier', 'Gradient Boosting Classi
df
```

	accuracy	TPR
Gradient Boosting Classifier	0.996308	0.970874
Gradient Boosting Classifier with CV	0.996308	0.970874

```
# Random Forest Classifier with CV
grid_values = {'max_features': ["auto"], 'min_samples_split': [2], 'n_estimators':
rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid=grid_values, cv=5,n_jobs=-1)
rf_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_features': ['auto'], 'min_samples_split': [2],
                         'n_estimators': [100]})
```

```
y_pred = rf_cv.predict(X_test)
acc_rf_cv = accuracy_score(y_test, y_pred)
acc_rf_cv
```

```
0.9932307692307693
```

```
tpr_rf_cv = recall_score(y_test, y_pred)
tpr_rf_cv
```

```
    0.8640776699029126
```

```
rf_cv.best_params_
```

```
    {'max_features': 'sqrt', 'min_samples_split': 50, 'n_estimators': 200}
```

```
df = pd.DataFrame(np.array([[acc_rf, tpr_rf], [acc_rf_cv, tpr_rf_cv]]),
                  columns=['accuracy', 'TPR'],
                  index=['Random Forest Classifier', 'Random Forest Classifier with
df
```

	accuracy	TPR
Random Forest Classifier	0.992308	0.834951
Random Forest Classifier with CV	0.993231	0.864078

```
grid_values = {'criterion': ["squared_error"]}
dtr = DecisionTreeRegressor()
dtr_cv = GridSearchCV(dtr, param_grid=grid_values, cv=5,n_jobs=-1)
dtr_cv.fit(X_train, y_train)
```

```
    GridSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_jobs=-1,
                 param_grid={'criterion': ['squared_error']})
```

```
y_pred = dtr_cv.predict(X_test)
acc_dtr_cv = accuracy_score(y_test, y_pred)
acc_dtr_cv
```

```
    0.9953846153846154
```

```
tpr_dtr_cv = recall_score(y_test, y_pred)
tpr_dtr_cv
```

```
    0.941747572815534
```

```
df = pd.DataFrame(np.array([[acc_dtr, tpr_dtr], [acc_dtr_cv, tpr_dtr_cv]]),
                  columns=['accuracy', 'TPR'],
                  index=['Decision Tree Regressor', 'Decision Tree Regressor with C
df
```

	accuracy	TPR
Decision Tree Regressor	0.995077	0.932039
Decision Tree Regressor with CV	0.995385	0.941748