# Machine Learning Application in Private Education Business


## IEOR 142 - Machine Learning & Data Analytics

Sunny Sun, Yinglu Deng, Cloris Zhang

# Table of Contents

# Goal and Motivation

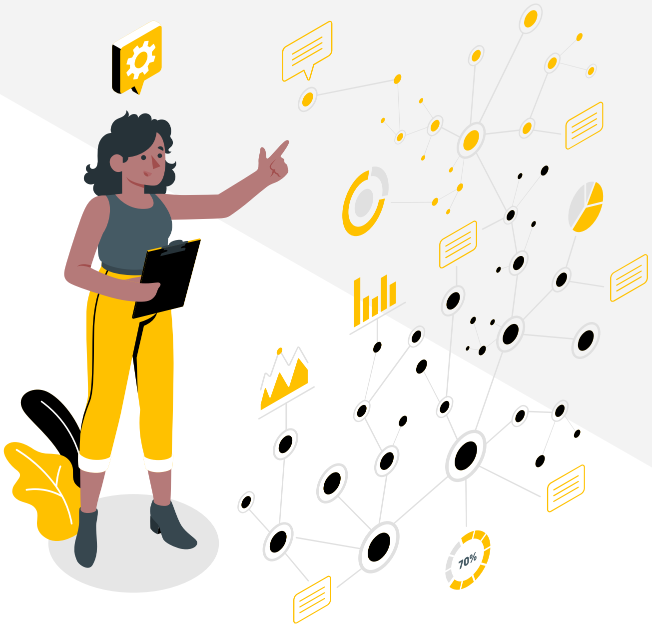
**Goal:**

1. Successful Sales Prediction
2. Sales Representative Performance

**Motivation:**

Increase sales, reduce cost & save time.

Increase effective customer check-ins, reduce data collection, customize the assignment of different customers to sales rep, and etc.



# Dataset

- **Sources:** Private Education Company's Customer Profile and Sales Data from Jan to Oct 2021.
- **Number of Columns:** 190
- **Number of Observations:** 10835 Customers
- **Number of Successful Customers:** 298
- **Total Sales:** ¥52, 107, 050

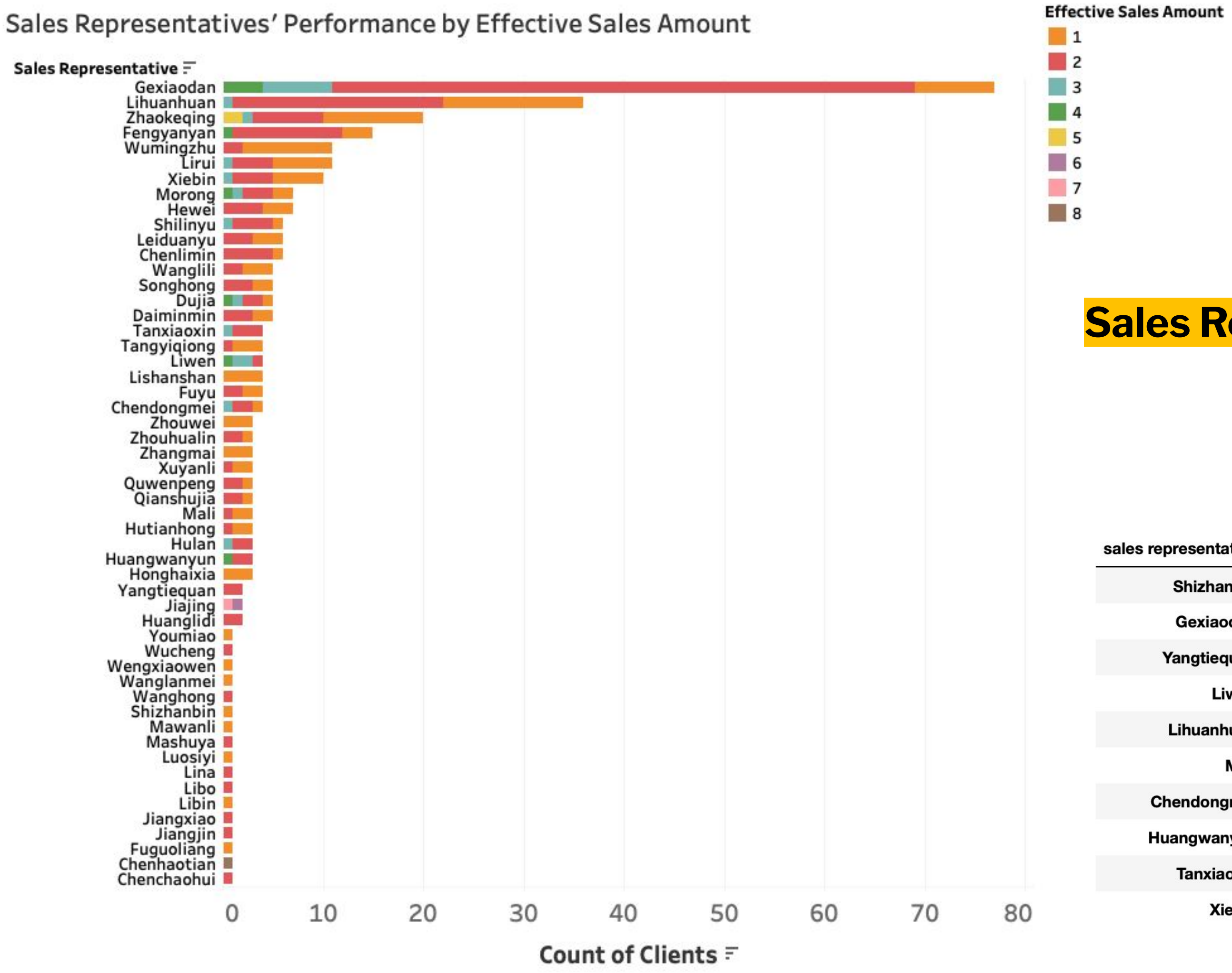
# Variables

- **(c) Customer Source:** Online Operations, Teaching Center, etc
- **(c)Status:** Collected, Self-Configured, etc
- **(c)Business Registration:** Yes, No, unknown
- **(c)Interest Category:** A~G (Different check-in frequency)
- **(c)Business Category:** Custom, Employed, Study Abroad, etc
- **(c)Customer Type:** Individual, Channel, Education Abroad, etc
- **(c)Customer Pool:** Entire, Entire-Collectible, East, East-1, etc
- **(c)Province:** Shanghai, Guangdong, Beijing, Xinjiang, etc
- **(c)City:** Shanghai, Beijing, Xiamen, Wuxi, etc
- **(n)Customer Update Status:** 1, 0
- **(n)Customer Rating:** 0~100 (Potential to purchase)
- **(n)Total Sales Amount:** Sales amount of signed contract
- **(n)Program Sales Total:** Total Price of Interested Program(s)
- **(n)Expected Sales:** Initial Price of Interested Program
- **(n)Number of Calls:** Number of Sale Calls
- **(d) Input Date:** Date of Data Input
- **(d) Recent Change Date:** Date of the most recent activity

Key:
(c) -character (n)-numeric (d)-datetime

# Data Cleaning/Feature Engineering

- Data types: Convert column types to appropriate data types
- Missing values: Replaced “null” with “unknown”
- Extract Actual Sales columns to avoid direct causal relationship with target variable(success)
- Delete columns with similar information & high correlation
  - Customer Rating & Interest Category
- Categorize specific programs names into general groups(MBA, EMBA, FDBA, etc)
- Calculate Deltatime for most recent date and input date




# Sales Representatives' Performance

| sales representative | # of successful customer | # of assigned customer | succ/assigned |
|---|---|---|---|
| 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 |
| Xiebin | 10 | 144 | 0.069444 |

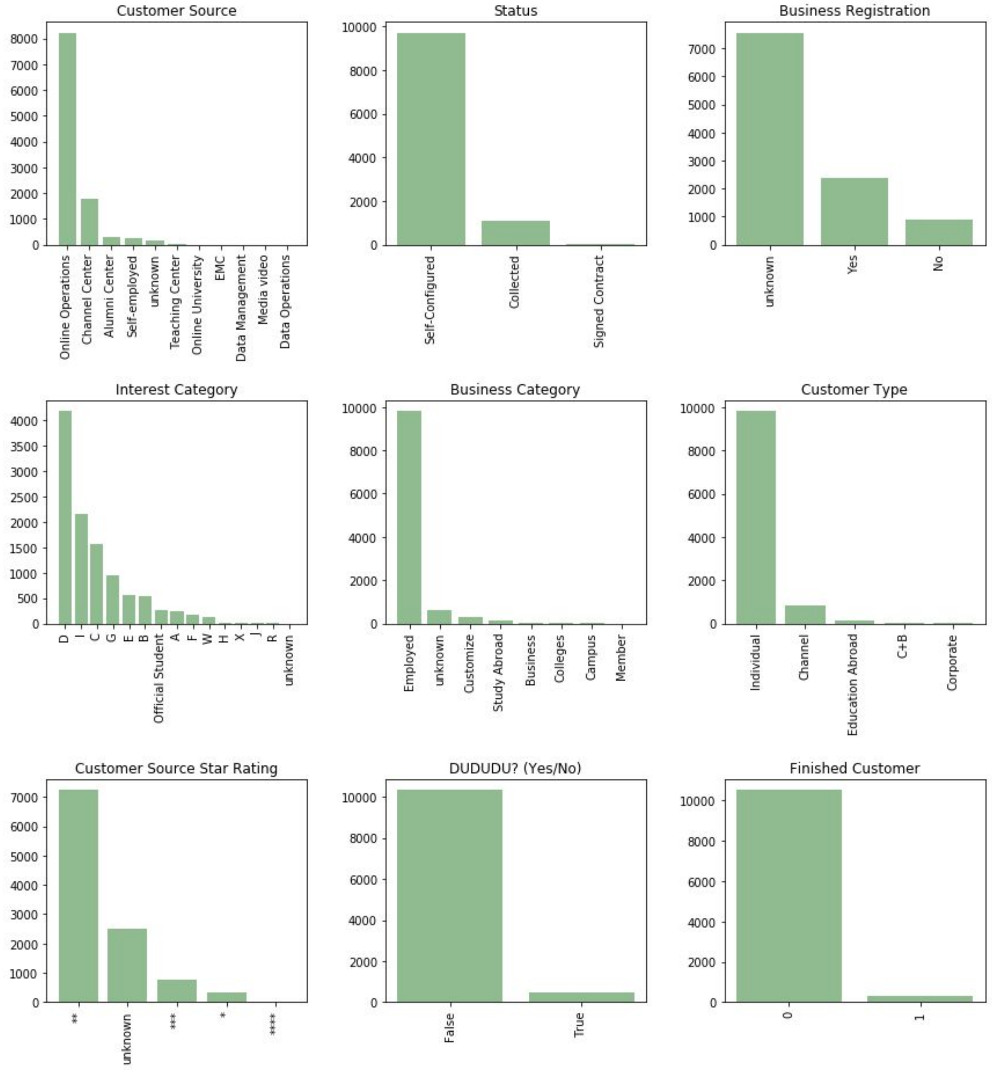
# Market Analysis

03

# MARKET ANALYSIS (EDA)

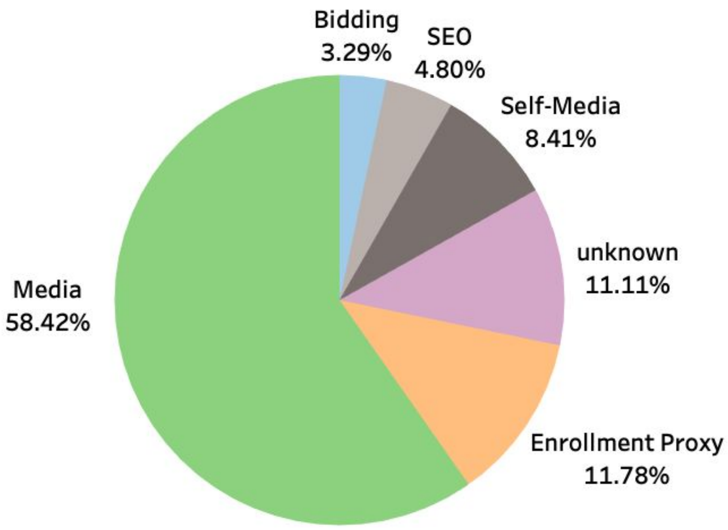
## 9 Important Categorical Features:

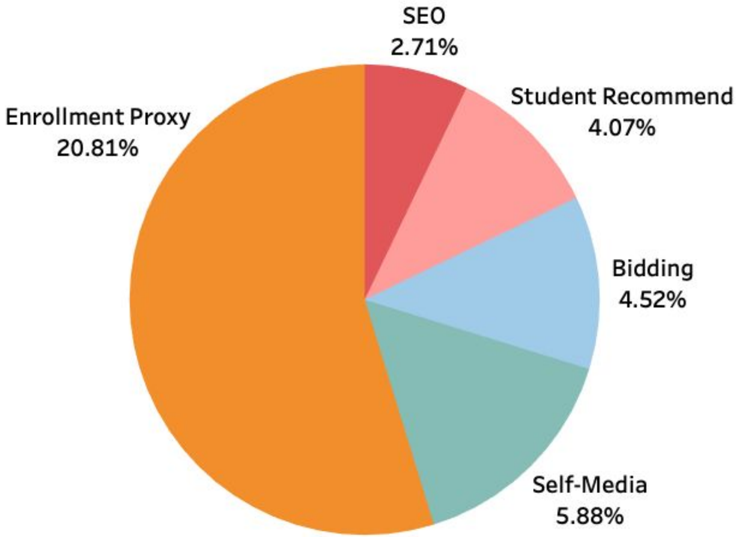
- The majority of customer sources are from online operations.
- Most of the status are self-configured.
- There are more businesses with registering.
- Most of the customers' type are individual.
- More two stars rating customer source.
- The number of finished customers is low which means we have a lot of potential customers.



# About Customer Source

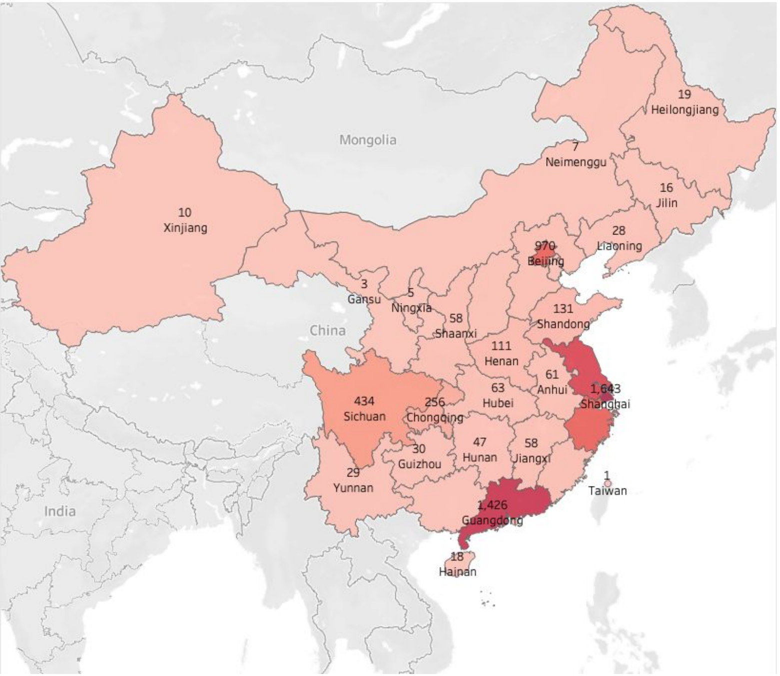
Clients Source from Whole Dataset **vs.** Successful Customers Source



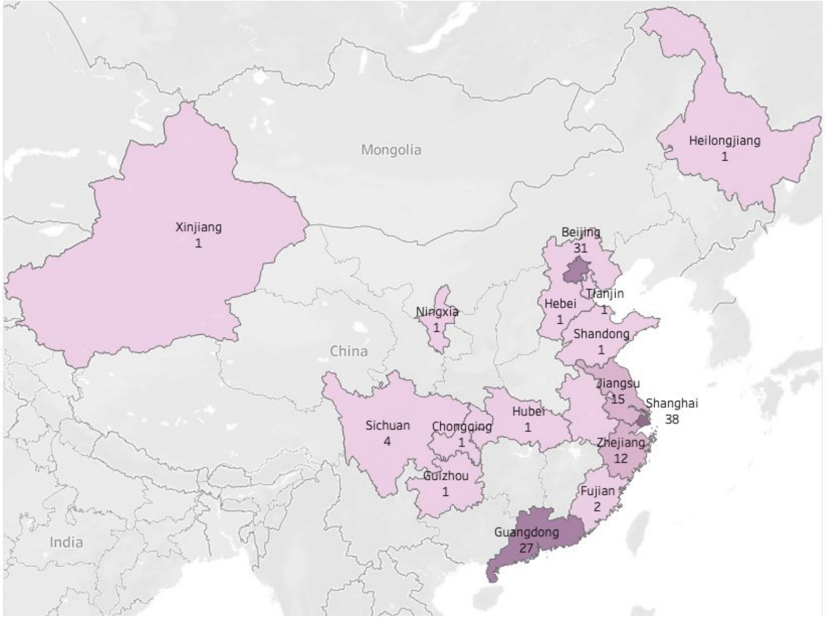




Clients Source by Medium (Secondary Classification)

## Clients' Source by Location



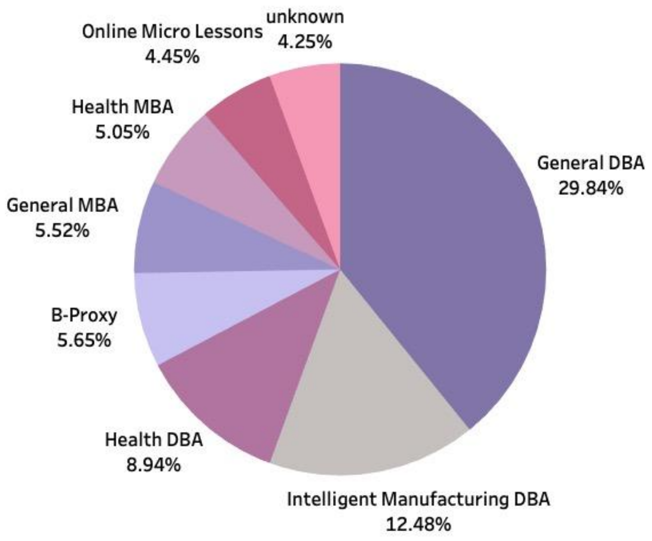

## Successful Clients' Location



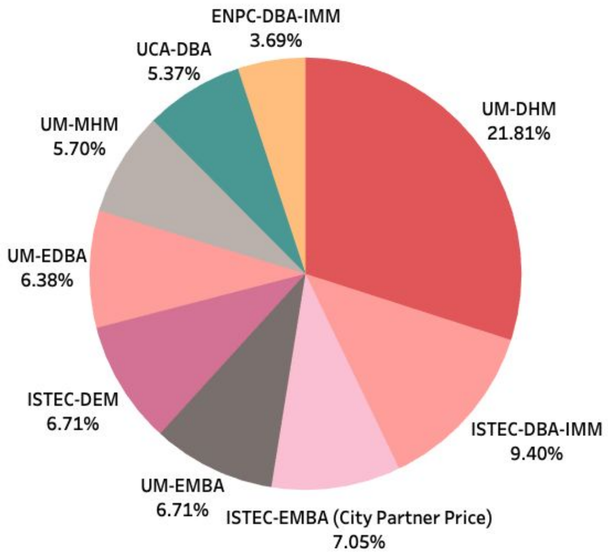

- Top 3 main clients source: Shanghai, Beijing, Guangdong.
- Potential customers are mainly from coastal areas such as Jiangsu, Shanghai, Zhejiang, Fujian, Guangdong.

# About Programs

- Total 80 different programs
- Top three most popular programs are all DBA
- However, the trending products are totally different (no DBA programs in the top 3)




The proportion of customers' interested program in the whole dataset

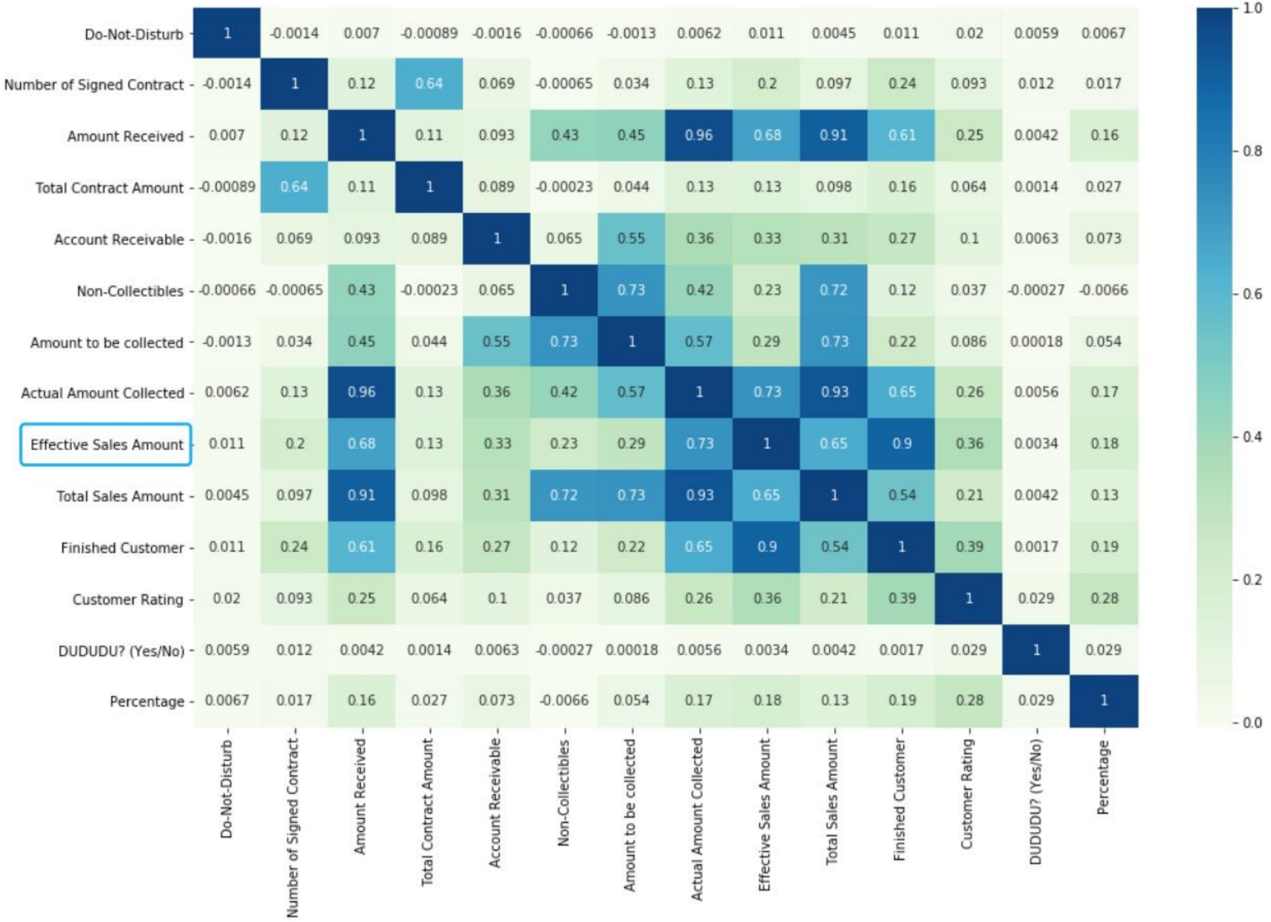



The proportion of success clients' program

# Correlation Matrix

**Findings:**

1. For effective sales amount, there are moderate correlations (r = 0.68, 0.73, 0.65) between the effective sales amount and amount received, actual amount collected, total sales amount respectively.
2. And there is a strong correlation (r = 0.9) between the effective sales amount and finished customer.



| | 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 | DUDUDU? (Yes/No) | Percentage |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Do-Not-Disturb | 1 | -0.0014 | 0.007 | -0.00089 | -0.0016 | -0.00066 | -0.0013 | 0.0062 | 0.011 | 0.0045 | 0.011 | 0.02 | 0.0059 | 0.0067 |
| Number of Signed Contract | -0.0014 | 1 | 0.12 | 0.64 | 0.069 | -0.00065 | 0.034 | 0.13 | 0.2 | 0.097 | 0.24 | 0.093 | 0.012 | 0.017 |
| Amount Received | 0.007 | 0.12 | 1 | 0.11 | 0.093 | 0.43 | 0.45 | 0.96 | 0.68 | 0.91 | 0.61 | 0.25 | 0.0042 | 0.16 |
| Total Contract Amount | -0.00089 | 0.64 | 0.11 | 1 | 0.089 | -0.00023 | 0.044 | 0.13 | 0.13 | 0.098 | 0.16 | 0.064 | 0.0014 | 0.027 |
| Account Receivable | -0.0016 | 0.069 | 0.093 | 0.089 | 1 | 0.065 | 0.55 | 0.36 | 0.33 | 0.31 | 0.27 | 0.1 | 0.0063 | 0.073 |
| Non-Collectibles | -0.00066 | -0.00065 | 0.43 | -0.00023 | 0.065 | 1 | 0.73 | 0.42 | 0.23 | 0.72 | 0.12 | 0.037 | -0.00027 | -0.0066 |
| Amount to be collected | -0.0013 | 0.034 | 0.45 | 0.044 | 0.55 | 0.73 | 1 | 0.57 | 0.29 | 0.73 | 0.22 | 0.086 | 0.00018 | 0.054 |
| Actual Amount Collected | 0.0062 | 0.13 | 0.96 | 0.13 | 0.36 | 0.42 | 0.57 | 1 | 0.73 | 0.93 | 0.65 | 0.26 | 0.0056 | 0.17 |
| Effective Sales Amount | 0.011 | 0.2 | 0.68 | 0.13 | 0.33 | 0.23 | 0.29 | 0.73 | 1 | 0.65 | 0.9 | 0.36 | 0.0034 | 0.18 |
| Total Sales Amount | 0.0045 | 0.097 | 0.91 | 0.098 | 0.31 | 0.72 | 0.73 | 0.93 | 0.65 | 1 | 0.54 | 0.21 | 0.0042 | 0.13 |
| Finished Customer | 0.011 | 0.24 | 0.61 | 0.16 | 0.27 | 0.12 | 0.22 | 0.65 | 0.9 | 0.54 | 1 | 0.39 | 0.0017 | 0.19 |
| Customer Rating | 0.02 | 0.093 | 0.25 | 0.064 | 0.1 | 0.037 | 0.086 | 0.26 | 0.36 | 0.21 | 0.39 | 1 | 0.029 | 0.28 |
| DUDUDU? (Yes/No) | 0.0059 | 0.012 | 0.0042 | 0.0014 | 0.0063 | -0.00027 | 0.00018 | 0.0056 | 0.0034 | 0.0042 | 0.0017 | 0.029 | 1 | 0.029 |
| Percentage | 0.0067 | 0.017 | 0.16 | 0.027 | 0.073 | -0.0066 | 0.054 | 0.17 | 0.18 | 0.13 | 0.19 | 0.28 | 0.029 | 1 |



Range: [-1, 1]

- No correlation r=0
- Very weak correlation: r<20
- Weak correlation: between 0.20-0.49
- Moderate correlation: between 0.5-0.79
- Strong correlation: between 0.8-0.99
- Perfect correlation: r=1

| | Number of Calls | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Effective Sales Amount | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 1 | 55 | 20 | 8 | 7 | 4 | 1 | 3 | 3 | 1 | | 2 | 1 | 1 | 1 |
| 2 | 101 | 21 | 16 | 8 | 6 | 3 | 5 | 2 | 1 | 2 | 1 | | 1 | |
| 3 | 13 | 2 | 3 | | 1 | | | | | | | | | |
| 4 | 9 | | | | | | | | | | | | | |
| 5 | 1 | 1 | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | 1 | |
| 8 | 1 | | | | | | | | | | | | | |

# Models

Select a list of base models (no hyperparameter tuning)

a. Decision Tree Classifier
b. Decision Tree Regressor
c. KNeighbors Classifier
d. Support Vector Machines Classifier
e. Multi-layer Perceptron Classifier
f. Linear Discriminant Analysis
g. Logistic Regression Classifier
h. Random Forest Classifier
i. Gradient Boosting Classifier

# Decision Tree Classifier

```
# 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_pred, y_test)
acc_dtc
```

```
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
```

# Decision Tree Regressor

```python
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
```

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

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| False | 1.00 | 1.00 | 1.00 | 3147 |
| True | 0.92 | 0.95 | 0.93 | 103 |
| accuracy | | | 1.00 | 3250 |
| macro avg | 0.96 | 0.97 | 0.97 | 3250 |
| weighted avg | 1.00 | 1.00 | 1.00 | 3250 |

# KNeighbors Classifier

```
# 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
```

```
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
```

# Support Vector Machines Classifier

```
# 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
```

```
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
```

# Multi-layer Perceptron Classifier

```
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
```

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

```
              precision    recall  f1-score   support

       False       0.97      1.00      0.98      3147
        True       0.44      0.07      0.12       103

    accuracy                           0.97      3250
   macro avg       0.70      0.53      0.55      3250
weighted avg       0.95      0.97      0.96      3250
```

# Linear Discriminant Analysis

```
# 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
```

```
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 |

# Logistic Regression Classifier

```python
# 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
```

```python
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
```

# Random Forest Classifier

```
# 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
```

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

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| False | 0.99 | 1.00 | 1.00 | 3147 |
| True | 0.93 | 0.84 | 0.88 | 103 |
| accuracy | | | 0.99 | 3250 |
| macro avg | 0.96 | 0.92 | 0.94 | 3250 |
| weighted avg | 0.99 | 0.99 | 0.99 | 3250 |

# Gradient Boosting Classifier

```
# 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
```

```
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 |

# Compare

- Accuracy
  - measure how often the algorithm classifies a data point correctly
- TPR: true positive rate / sensitivity / recall
  - measure the percentage of actual positives which are correctly identified

| | 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 |

# Hyperparameter Tuning

- Random Forest
- TPR increases 0.03

```
# Random Forest Classifier with CV
grid_values = {'max_features': ["auto"], 'min_samples_split': [2], 'n_estimators': [100]}
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
```

```
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 CV'])
df
```

| | accuracy | TPR |
|---|---|---|
| **Random Forest Classifier** | 0.992308 | 0.834951 |
| **Random Forest Classifier with CV** | 0.993231 | 0.864078 |

# Hyperparameter Tuning

- Decision Tree Regressor
- TPR increase 0.01

```
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
```

```
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 CV'])
df
```

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

# Hyperparameter Tuning

- Gradient Boosting

```
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
```

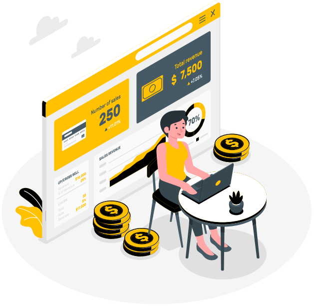
```
0.9963076923076923
```

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

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

# Next Steps

- Do more cross-validation
- Try other machine learning models
- Use NLP to extract key information from Remarks Section to improve customer profile
- Conduct Feature_Importance to find key variables to reduce data collection process.




05

# Thanks !