Bank Marketing

Strategy & Tactical Plan

Yinglu Deng, Anji Dong, Cloris Zhang, Sunny Sun

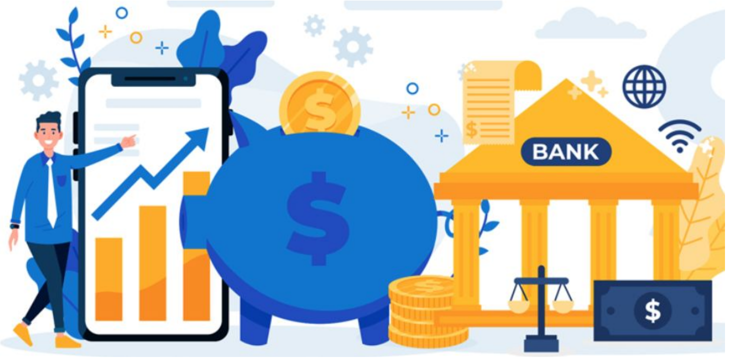
Introduction: Project Context

Research Question:

What are important factors that contribute to a customer's decision on deposit?

Motivation:

Higher efficiency in aiming for customers (ex: advertise more to specific targets)

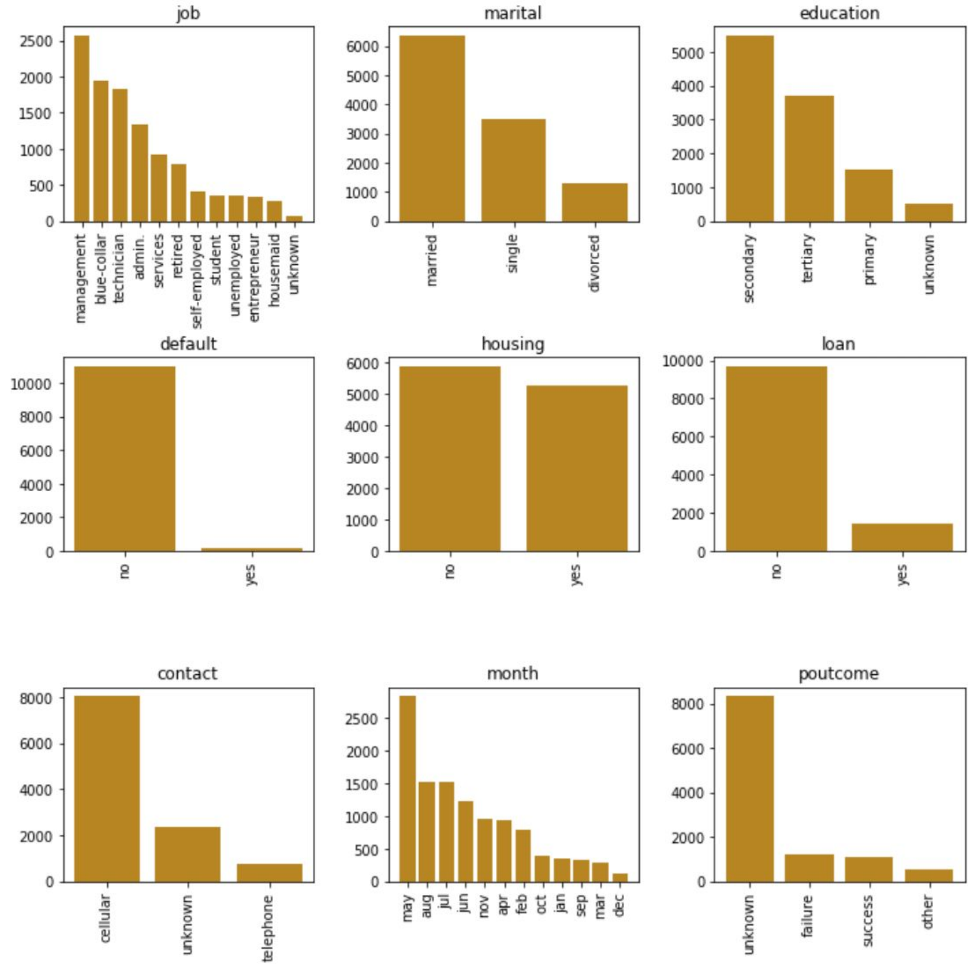


Bank Marketing Client Dataset

- From UCI Machine Learning Repository
  - Train.csv - 8,929 rows
  - Test.csv - 2,233 rows
- 9 categorical features and 7 numerical features
- Target: ”deposit” column (has the client subscribed a term deposit?)

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

9 Categorical Features:



Bar plot findings:

1. Top three job field: Management; blue-collar; technician

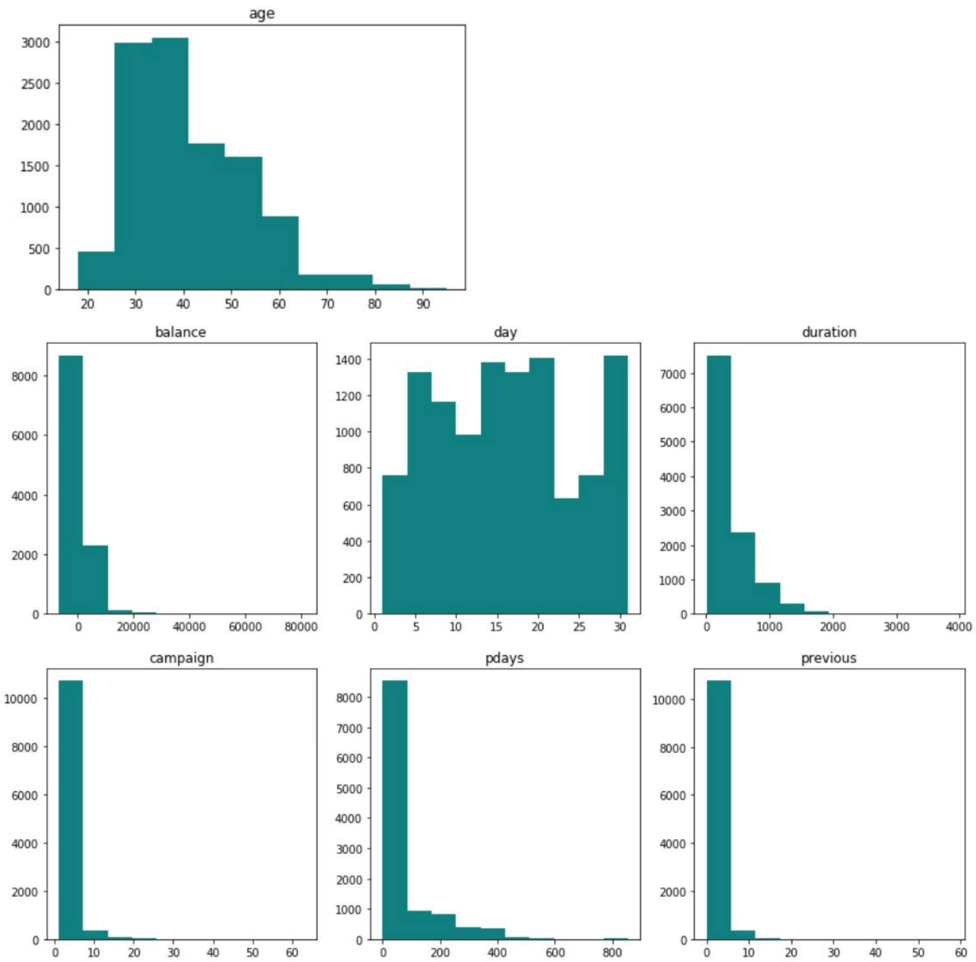
2. Most of clients are married and with higher education (secondary and tertiary), no credit in default, don't have personal loan.

3. The majority communication type is cellular.

4. Last contact months are mainly focused on May, June, July and August.

5. There are slightly higher amount of failure than success in the outcome of the previous marketing campaign.

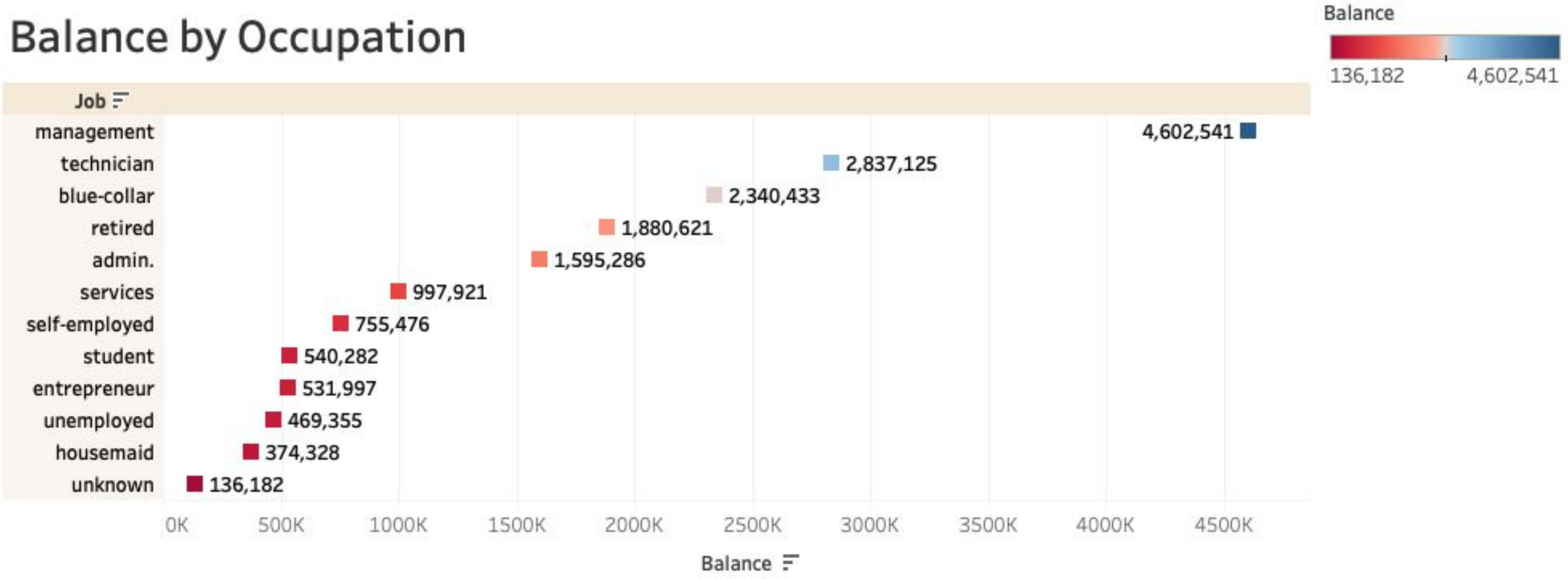
7 Numerical Features:



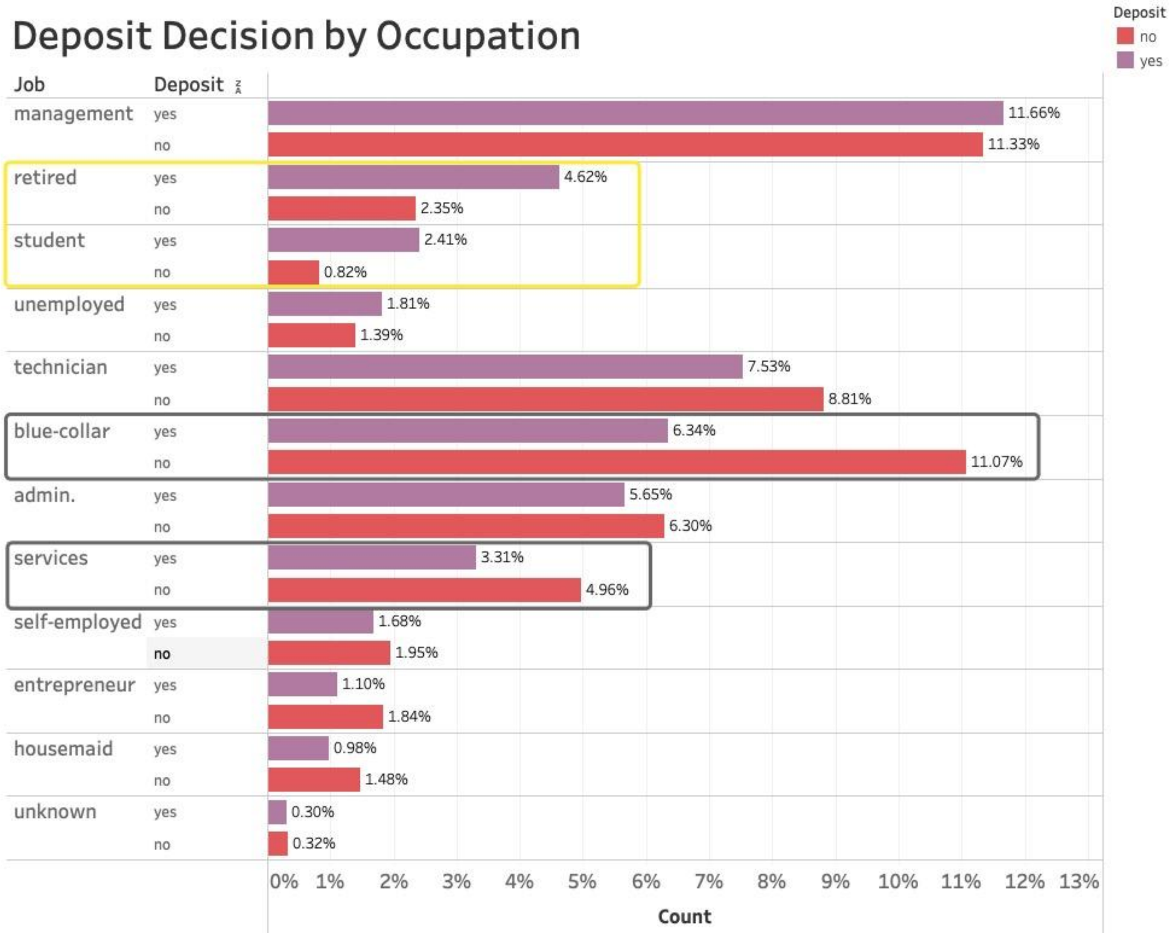
Histograms findings:

1. Medium age of our clients are around 30 - 50 years old.
2. Most of the numerical columns are not normally distributed and some of them have outliers.

Data Analysis:



- Management, technician and blue-collar are the ones who have the highest balance in their accounts.

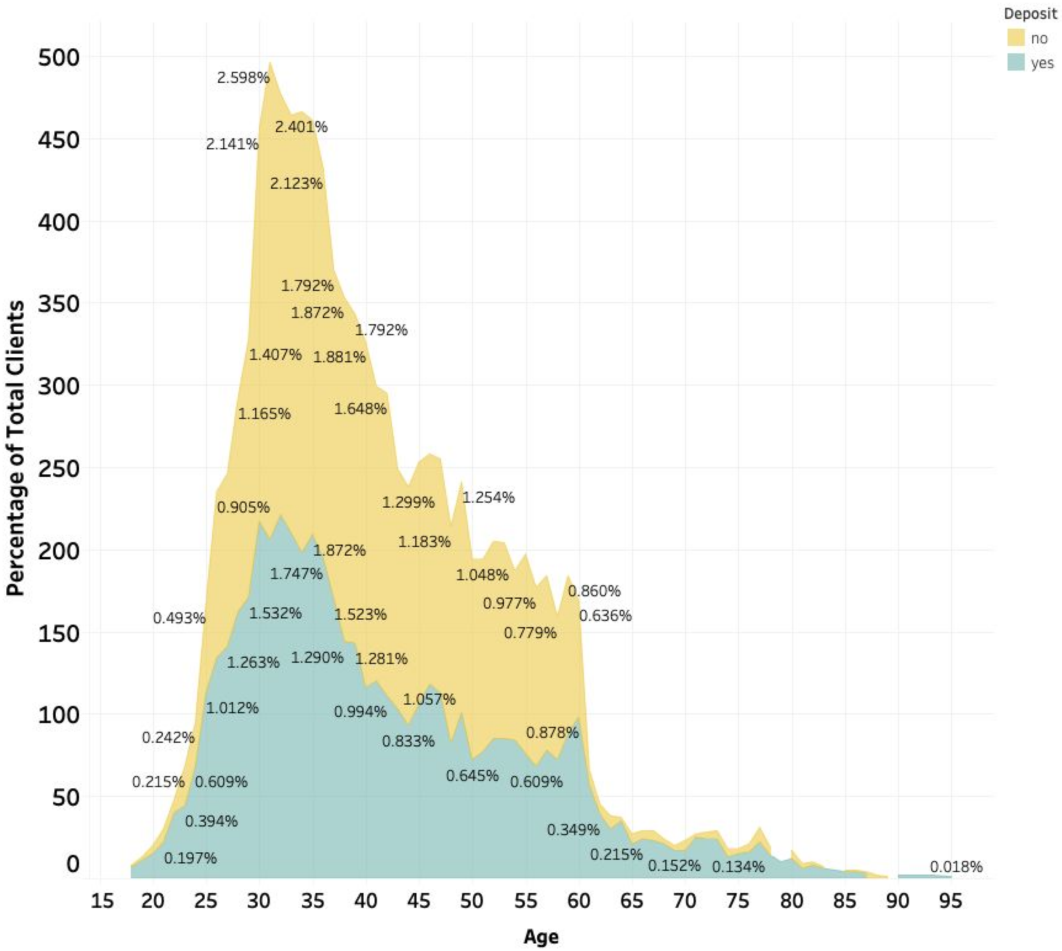


- Retired clients and students are more likely to subscribe a term deposit.
- Customers with blue-collar and services' jobs are less likely to subscribe.

The **characteristic** of clients who are more likely to subscribed for term deposit:

1. No housing loan
2. Success outcome of previous marketing campaign
3. By cellular contact communication
4. Married status
5. Higher level education

					Education			
Deposit	Housing	Poutcome	Contact	Marital	primary	secondary	tertiary	unknown
yes	no	success	cellular	divorced	6	25	21	3
				married	32	171	143	17
				single	7	83	139	15
			telephone	divorced	6	1		1
				married	13	17	9	7
				single	1	7	4	
		failure	cellular	divorced	2	11	7	3
				married	17	70	71	9
				single	3	37	60	7
			telephone	married	10	9	3	3
				single		3	3	
	yes	success	cellular	divorced	4	13	10	
				married	11	72	37	11
				single		35	44	1
			telephone	divorced		2		
				married	1	4	1	
				single			1	
		failure	cellular	divorced	3	16	10	1
				married	19	86	38	1
				single	2	44	54	2
			telephone	divorced		1		1
				married	1	3	2	
				single		1	1	



- The number of people who are 25 to 45 years old with a term deposit account is high.

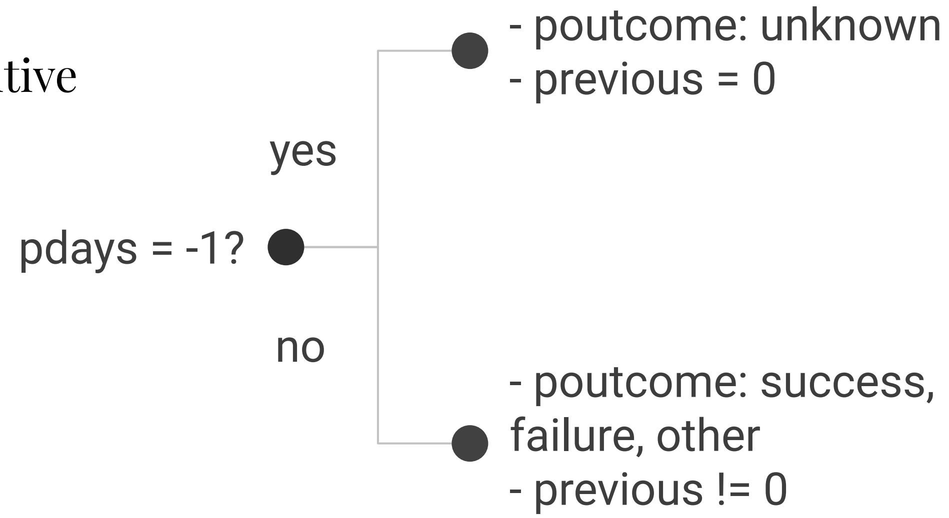
Correlation Matrix

There is a moderate correlation (r = 0.51) between the days and the previous days.

Data Cleaning & Feature Engineering

1. Inconsistency & Errors

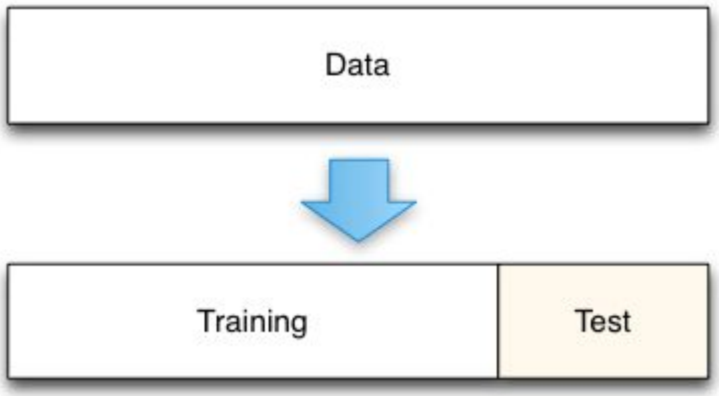
- Check NAN values
- Check upper/lower case sensitive
- Check consistency



"pdays": number of days that passed by after the contact from previous campaign (-1 if not previously contacted)
"poutcome": outcome for the previous campaign (success, failure, other, unknown)
"previous": number of contacts for previous campaign

Train Test Split

- Training set: 70%
- Testing set: 30%




```
y = df["deposit"]
X = df.drop(['deposit'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

2. Convert Ordinal Categorical variables

"Education"

- primary, secondary, tertiary, unknown
- Remove "unknown"
- {primary: 1, secondary: 2, tertiary: 3}

```
# mode of education for each job title
dictionary = {}
for job in df['job'].unique():
    accounts_with_job = df[df['job']==job]
    mode_job = accounts_with_job['education'].mode()[0]
    dictionary[job] = mode_job
```

3. Check Outliers for numerical variables

- Find the outlier range (Q1-1.5*IQR, Q3+1.5*IQR)
- % of outliers > 5% : “balance”, “duration”, “campaign”, “pdays”, “previous”
- Significant ones: “pdays”, “previous”, “balance”
- Solutions
  - Lower bound and upper bound (Q1-2.5*IQR, Q3+2.5*IQR)

```
The percentage of outliers in 'balance' column is 9.50979137335210 5%.
The largest outlier is 49.17622689611217 6 IQR above/below the outlier range.

The percentage of outliers in 'pdays' column is 17.97004991680532 6%.
The largest outlier is 16.931818181818183 IQR above/below the outlier range.

The percentage of outliers in 'previous' column is 11.263279150134393%.
The largest outlier is 55.5 IQR above/below the outlier range.
```

“Pdays” & “previous”

- ~ 74% of Newly contacted customers (pdays=-1 and previous=0)
- Split the data into newly contacted and previously contacted
- Added a new binary feature “not_previously_contacted”

```
% outliers in 'pdays' column for previously contacted customers is 0.307180340458210 7%.
% outliers in 'previous' column for previously contacted customers is 11.26327915013439 3%.
The largest outlier is 0.9367816091954023 IQR above/below the outlier range.
```

4. Normalization & dummy variables

- Used StandardScaler() to normalize numerical values
- create dummy variables for categorical variables

```
X_train.head()
```

	age	balance	day	duration	campaign	pdays	previous
0	2.765525	–0.755311	–0.324645	0.631535	–0.188574	–0.489456	–0.358947
1	1.168241	–0.724987	–1.038703	0.415362	0.919609	–0.489456	–0.358947
2	1.336376	–0.598118	1.341489	–0.497015	–0.742666	–0.489456	–0.358947
3	–0.344975	1.704700	–0.562664	–0.636892	–0.188574	–0.489456	–0.358947
4	1.252309	2.719648	1.579509	–0.729083	–0.742666	1.330014	0.067780

7813 rows × 43 columns

$$Z = \frac{x - \mu}{\sigma}$$

Methods and Approaches

Select a list of base models (no hyperparameter tuni

a. Decision Tree Classifier
b. KNeighbors Classifier
c. Support Vector Machines Classifier
d. Multi-layer Perceptron classifier
e. Linear Discriminant Analysis
f. Logistic Regression Classifier
g. Random Forest Classifier
h. 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_test, y_pred)
acc_dtc
```

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

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79      1760
           1       0.77      0.76      0.76      1589

    accuracy                           0.78      3349
   macro avg       0.78      0.77      0.77      3349
weighted avg       0.78      0.78      0.78      3349
```

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
```

```
0.8154673036727381
```

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

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.83      1760
           1       0.82      0.78      0.80      1589

    accuracy                           0.82      3349
   macro avg       0.82      0.81      0.81      3349
weighted avg       0.82      0.82      0.82      3349
```

Support Vector Machines Classifier

```python
# 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
```

```
0.8507017020005972
```

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

```
              precision    recall  f1-score   support

           0       0.88      0.83      0.85      1760
           1       0.82      0.88      0.85      1589

    accuracy                           0.85      3349
   macro avg       0.85      0.85      0.85      3349
weighted avg       0.85      0.85      0.85      3349
```

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
```

```
0.8450283666766198
```

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

	precision	recall	f1-score	support
0	0.87	0.83	0.85	1760
1	0.82	0.86	0.84	1589
accuracy			0.85	3349
macro avg	0.84	0.85	0.84	3349
weighted avg	0.85	0.85	0.85	3349

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
```

```
0.8256195879366975
```

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

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84      1760
           1       0.85      0.77      0.81      1589

    accuracy                           0.83      3349
   macro avg       0.83      0.82      0.82      3349
weighted avg       0.83      0.83      0.82      3349
```

Logistic Regression Classifier

```
# 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
```

```
0.826515377724694
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      1760
           1       0.83      0.79      0.81      1589

    accuracy                           0.83      3349
   macro avg       0.83      0.82      0.83      3349
weighted avg       0.83      0.83      0.83      3349
```

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
rf_pred = rf_model.predict(X_test)
# model Evaluation
acc_rf = accuracy_score(y_test, rf_pred)
acc_rf
```

```
0.8504031054045984
```

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

	precision	recall	f1-score	support
0	0.82	0.86	0.84	1760
1	0.83	0.79	0.81	1589
accuracy			0.83	3349
macro avg	0.83	0.82	0.83	3349
weighted avg	0.83	0.83	0.83	3349

Gradient Boosting Classifier

```python
# 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
```

```
0.8414452075246343
```

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

	precision	recall	f1-score	support
0	0.86	0.83	0.85	1760
1	0.82	0.85	0.84	1589
accuracy			0.84	3349
macro avg	0.84	0.84	0.84	3349
weighted avg	0.84	0.84	0.84	3349

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	0.775754	0.755192
KNeighbors Classifier	0.815467	0.784770
Support Vector Machines Classifier	0.850702	0.877281
Multi-layer Perceptron Classifier	0.845028	0.858402
Linear Discriminant Analysis	0.825620	0.770296
Logistic Regression	0.826515	0.794210
Random Forest Classifier	0.850702	0.880428
Gradient Boosting	0.841445	0.852738

```
df = pd.DataFrame(np.array([[acc_dtc, tpr_dtc], [acc_knn, tpr_knn], [acc_svm, tpr_svm], [acc_mlp, tpr_mlp], [acc_lda, tpr_lda],
                            [acc_log, tpr_log], [acc_rf, tpr_rf], [acc_gbc, tpr_gbc]]),
                  columns=['accuracy', 'TPR'],
                  index=['Decision Tree Classifier', 'KNeighbors Classifier', 'Support Vector Machines Classifier', 'Multi-layer Perceptron Classifier',
                  'Linear Discriminant Analysis', 'Logistic Regression', 'Random Forest Classifier', 'Gradient Boosting'])
df
```

Hyperparameter Tuning

- GridSearchCV
  - helps to loop through predefined hyperparameters and fit the estimator (model) on training set
  - in the end, we can select the best parameters from the listed hyperparameters
- e.g. GradientBoostingClassifier

```
param_grid = {'loss': ['exponential', 'deviance'], 'learning_rate' : [0.001, 0.01, 0.1, 1, 10, 100], 'n_estimators': [50, 100, 500]}
gbc = GridSearchCV(GradientBoostingClassifier(), param_grid, verbose = -1)

# fitting the model for grid search
gbc.fit(X_train, y_train)
# print best parameter after tuning
print(gbc.best_params_)

# print how our model looks after hyper-parameter tuning
print(gbc.best_estimator_)
```

```
{'learning_rate': 0.1, 'loss': 'deviance', 'n_estimators': 500}
GradientBoostingClassifier(n_estimators=500)
```

Hyperparameter Tuning

- Accuracy increases 0.01
- TPR increases 0.02

```
y_pred = gbc.predict(X_test)
# model Evaluation
acc_tune = accuracy_score(y_test, y_pred)
acc_tune
```

```
0.8560764407285757
```

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

```
0.8741346758967904
```

```
df = pd.DataFrame(np.array([[acc_gbc, tpr_gbc], [acc_tune, tpr_tune]]),
                  columns=['accuracy', 'TPR'],
                  index=['GradientB oosting Classifier', 'Gradient Boosting Classifier Tuned'])
df
```

	accuracy	TPR
GradientB oosting Classifier	0.841445	0.852738
Gradient Boosting Classifier Tuned	0.856076	0.874135

Hyperparameter Tuning

- MLPClassifier
  - activation, solver - default
  - learning_rate_init - 0.0005
  - batch size - 32
  - hidden_layer_sizes=(5, 5, 5)
- Accuracy increases 0.003
- TPR increases 0.03

	accuracy	TPR
Multi-layer Perceptron Classifier	0.845028	0.858402
Multi-layer Perceptron Classifier Tuned	0.848313	0.885463

```
] mlp = MLPClassifier(hidden_layer_sizes=(5, 5, 5), batch_size=(32), learning_rate_init=0.0005, activation='relu',solver='adam', verbose=0)

mlp.fit(X_train, y_train)
# Prediction
y_pred = mlp.predict(X_test)
# model Evaluation
acc_mlp_tune = accuracy_score(y_test, y_pred)
acc_mlp_tune
```

```
0.8483129292326067
```

Hyperparameter Tuning

- Support Vector Machines Classifier
  - PCA to reduce dimensionality
    - reduced accuracy
  - GridSearchCV to tune parameters like {‘C’, ‘gamma’, ‘kernel’}
    - accuracy increased by less than 0.5%.

```
#Trying to reduce the dimensionality for SVM classifier

from sklearn.decomposition import PCA
for i in np.arange(10,40):
    pca = PCA(n_components = i)
    pca.fit(X_train)
    pca2 = PCA(n_components = i)
    pca2.fit(X_test)
    X_test_pca = pca2.transform(X_test)
    X_pca = pca.transform(X_train)
    svm_pca = svm.SVC().fit(X_pca, y_train)
    svm_pred = svm_pca.predict(X_test_pca)

    # model Evaluation
    print("Accuracy PCA",i,": ",accuracy_score(y_test, svm_pred))
```

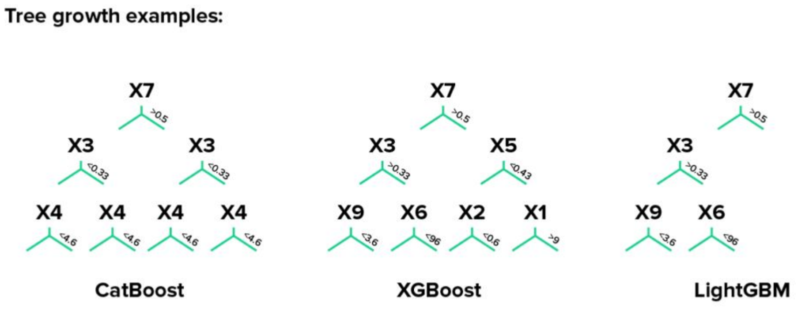
Hyperparameter Tuning

- Random Forest Classifier
  - 5-Fold CV
    - accuracy increased by less than 0.5%.
  - GridSearchCV to tune parameters like {'ccp_alpha'}
    - similar results

```
from sklearn.ensemble import RandomForestClassifier
grid_values = {'ccp_alpha' : [0,0.0001]}

rf_model = RandomForestClassifier(n_estimators=100)
cv = KFold(n_splits = 5, random_state = 1, shuffle = True)
rf_cv = GridSearchCV(rf_model,param_grid = grid_values, scoring = 'accuracy', cv = cv, verbose = 0)
# Fitting the classifier
rf_cv.fit(X_train, y_train)
rf_pred = rf_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, rf_pred))
#  Confusion matrix
print(confusion_matrix(y_test, rf_pred))
print(rf_cv.best_params_)
```

CatBoostingClassifier & Regressor

Yandex
CatBoost



- Algorithm for gradient boosting on Decision Tree
- Each successive tree is built with reduced loss compared to previous trees.
- Automatic Overfitting Detector
  - IncToDec: Threshold value in starting parameters > CurrentPV Value
  - Iter: # of iterations > value specified in training parameters

Source : https://catboost.ai/en/docs/concepts/overfitting-detector#iter

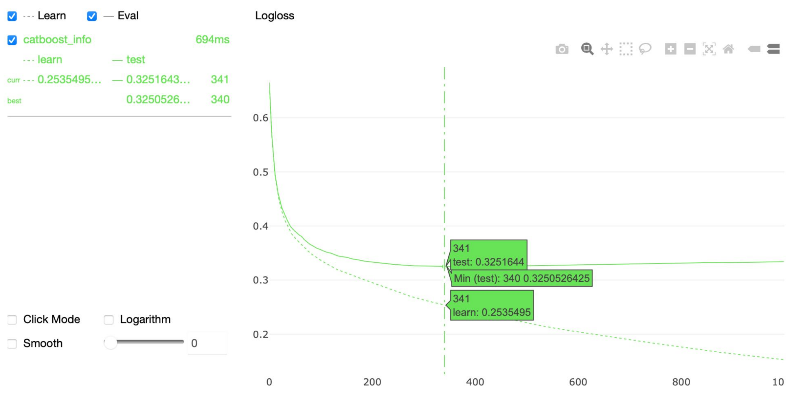
CatBoostClassifier

```
# Declaring classifier model
cbc = CatBoostClassifier()

# Fitting classifer to training set
cbc.fit(X_train, y_train,
          eval_set=(X_test, y_test),
          plot=True,use_best_model=True,
);

# Predicting test set
cbc_predict = cbc.predict(X_test)
```

- Baseline CatBoostClassifer model achieved a high accuracy of 85.8%.

```
CatBoostClassifier Report:

              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1760
           1       0.83      0.88      0.85      1589

    accuracy                           0.86      3349
   macro avg       0.86      0.86      0.86      3349
weighted avg       0.86      0.86      0.86      3349

Accuracy of CatBoostClassifier is: 0.858166169005673
```

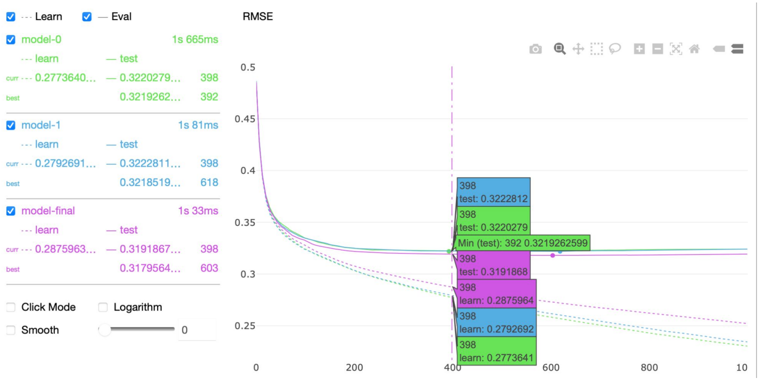
CatBoostClassifier with Tuning

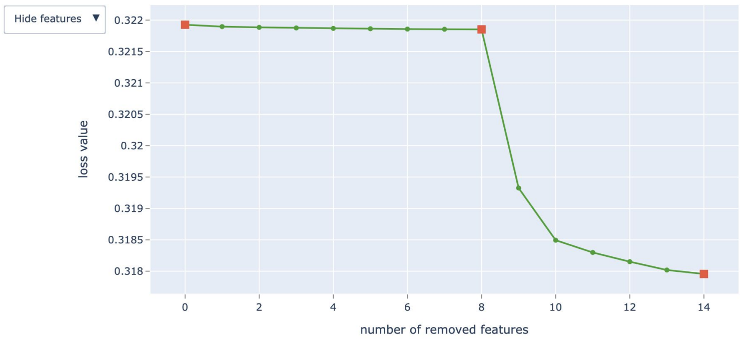



```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1760
           1       0.83      0.88      0.86      1589

    accuracy                           0.86      3349
   macro avg       0.86      0.86      0.86      3349
weighted avg       0.86      0.86      0.86      3349

Accuracy: 0.8590624066885637
[[1483  277]
 [ 195 1394]]
```

- Conducted RandomizedSearchCV to tune learning_rate and max_depth.
- RandomizedSearchCV requires less runtime than GridSearchCV while exploring same parameters and achieving similar performance.
- CatBoostClassifier (w/learning_rate = 0.5 & max_depth = 6) achieved 85.9% accuracy.
    - Slight improvement in both accuracy and f1-score for people with deposit.

CatBoostRegressor with Tuning

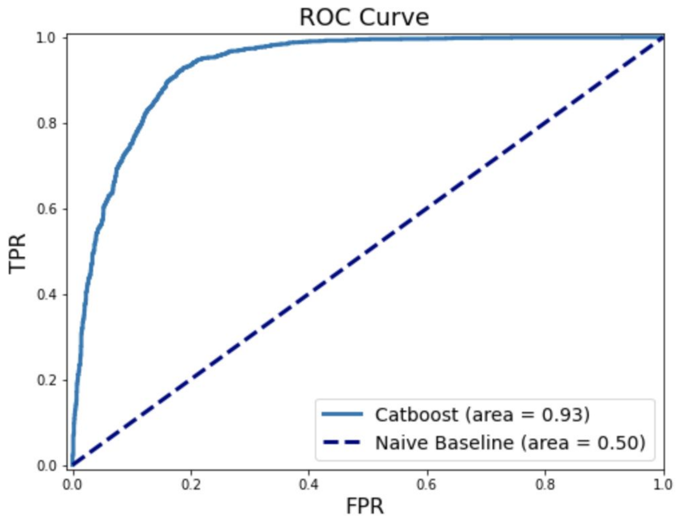




- Conducted select_features to find right amount of significant columns to achieve the lowest RMSE.
  - Initially starting with column counts from 20 to 40 and narrowed to 29
  - Used RecursiveByShapValues as the algorithm - most accurate method
- Selected columns are

```
Index(['day', 'duration', 'campaign', 'pdays', 'previous',
       'not_previously_contacted', 'job_housemaid', 'job_student',
       'marital_married', 'marital_single', 'education_3', 'default_yes',
       'housing_yes', 'loan_yes', 'contact_telephone', 'contact_unknown',
       'month_aug', 'month_dec', 'month_feb', 'month_jan', 'month_jul',
       'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct',
       'month_sep', 'poutcome_success', 'poutcome_unknown'],
      dtype='object')
```

CatBoostRegressor with Tuning (cont.)




```
# calculate the g-mean for each threshold
gmeans = np.sqrt(tpr * (1-fpr))
# locate the index of the largest g-mean
ix = np.argmax(gmeans)
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix], gmeans[ix]))
```

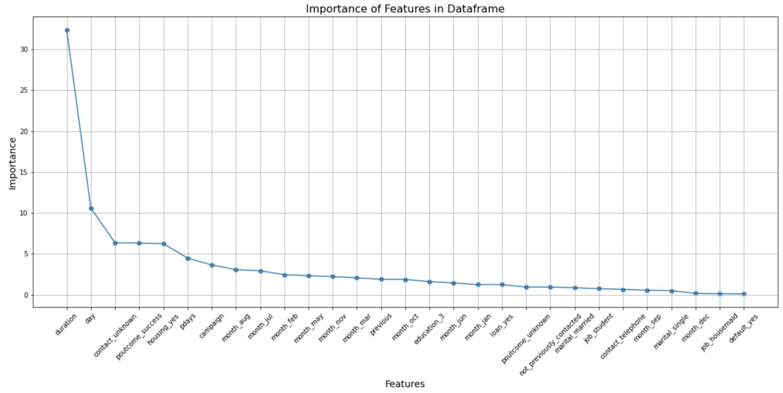
```
Best Threshold=0.456248, G-Mean=0.869
```

- Fitted the model with training sets of selected_features.
- High AUC ⇨ High model performance in distinguishing classes.
- Used G-Means to calculate the best threshold to cut off 0 and 1 for deposit.
    - Best Threshold = 0.456248
    - G-Mean=0.869

Looking at Feature Importance

- Conducting feature_importances_, we find the importances of the 29 variables we have selected.
- Top five features are duration, day, contact_unknown, poutcome_sucess, and housing_yes.
- Experimented with re-fitting the model without variables that have low importances (<= 1), accuracy dropped.

	Feature Id	Importances
0	duration	32.392715
1	day	10.595458
2	contact_unknown	6.353713
3	poutcome_success	6.321325
4	housing_yes	6.242035
5	pdays	4.482228
6	campaign	3.645062
7	month_aug	3.068043
8	month_jul	2.934039
9	month_feb	2.455316
19	poutcome_unknown	0.947882
20	not_previously_contacted	0.940161
21	marital_married	0.853954
22	job_student	0.752955
23	contact_telephone	0.657187
24	month_sep	0.549721
25	marital_single	0.493954
26	month_dec	0.170294
27	job_housemaid	0.126064
28	default_yes	0.108430



Final CatBoostRegressor

- Final CatBoostRegressor with 29 selected features.
- 87% Accuracy
- 0.92 Recall Rate (TPR)
- Correctly predicting customers that subscribed to term deposits.

	precision	recall	f1-score	support
0	0.92	0.82	0.87	1760
1	0.82	0.92	0.87	1589
accuracy			0.87	3349
macro avg	0.87	0.87	0.87	3349
weighted avg	0.87	0.87	0.87	3349

Findings

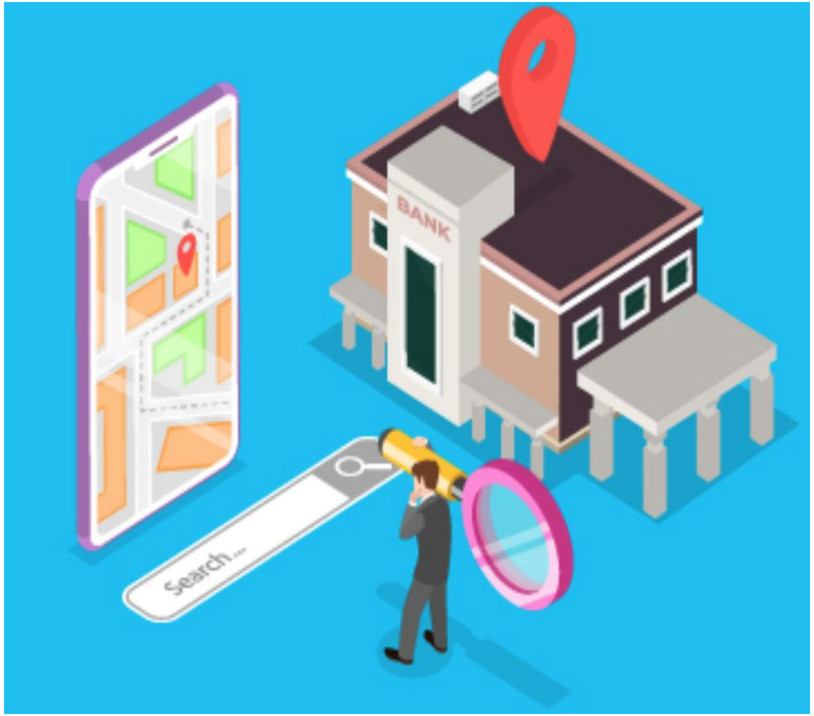
- Different approaches generated different performance metrics
- GridSearchCV and RandomizedSearchCV are useful for tuning hyperparameters
- CV improves TPR

Real World Application

- Marketing Campaign
    - target desired audience

Future Work

- Use more cross validation methods
- Learn and use some computational expensive machine learning algorithms
- Get more data that has valuable information



Thank you!

Questions?