Targeted Marketing Campaigns for Banking Clients Using Data Science

Executive Summary

Marketing Campaigns need to be designed around what objectives are required by the business.

Marketing helps build strong and new customer relationships. Businesses can leverage data science to accurately promote and sell products or services.

Using data science and machine learning teams can deliver value for targeted prospects and consumers. Catering content, with the long-term goal of demonstrating product value and strengthening brand loyalty will ultimately lead to increased revenue.

The goal of this analysis is to develop two strong models to predict which banking clients are more likely to sign up for a term deposit. Using marketing campaigns on marketing qualified leads (MQLs) and sales qualified leads (SQLs) will help increase sales velocity and revenue.

Report Outline

Importing dependencies

This project uses the classification models Gaussian Naive Bayes and Random Forest Classifier from sklearn. Matplotlib and seaborn were used for visualizations.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#import visualization tools
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

#import machine learning
import sklearn
from sklearn.ensemble import RandomForestClassifier,
from sklearn.naive_bayes import GaussianNB

#split
from sklearn.model_selection import train_test_split
#metrics
from sklearn.metrics import accuracy_score
```

Data Cleaning + Feature Engineering

We can start by cleaning the numeric variables and viewing the basic statistics.

```
#import our dataset and print the first 5 rows
data = pd.read_csv('../input/bank-marketing-dataset/bank.csv')

data.head()
```

	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

```
#explore last 5 rows
data.tail()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	20	apr	257	1	-1	0	unknown	no
11158	39	services	married	secondary	no	733	no	no	unknown	16	jun	83	4	-1	0	unknown	no
11159	32	technician	single	secondary	no	29	no	no	cellular	19	aug	156	2	-1	0	unknown	no
11160	43	technician	married	secondary	no	0	no	yes	cellular	8	may	9	2	172	5	failure	no
11161	34	technician	married	secondary	no	0	no	no	cellular	9	jul	628	1	-1	0	unknown	no

Exploratory Data Analysis (EDA)

```
#check how many rows are in the dataset
print('There are {rows} rows in the dataset.'.format(rows = len(data)))
```

The output of this cell results in the number of rows (here there are 11162).

```
#let's get a description of our data set
data.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330407	0.832557
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758282	2.292007
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000	0.000000
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000	0.000000
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000	0.000000
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000	1.000000
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000	58.000000

Summary of Numerical Data

The mean age of banking clients is 41 years old, minimum age is 18 years and maximum age is 95 years old.

The mean balance in each account is approximately $1528.54.

The standard deviation (STD) of the balance is 3225.41. The high STD indicates a wider spread of balances among clients.

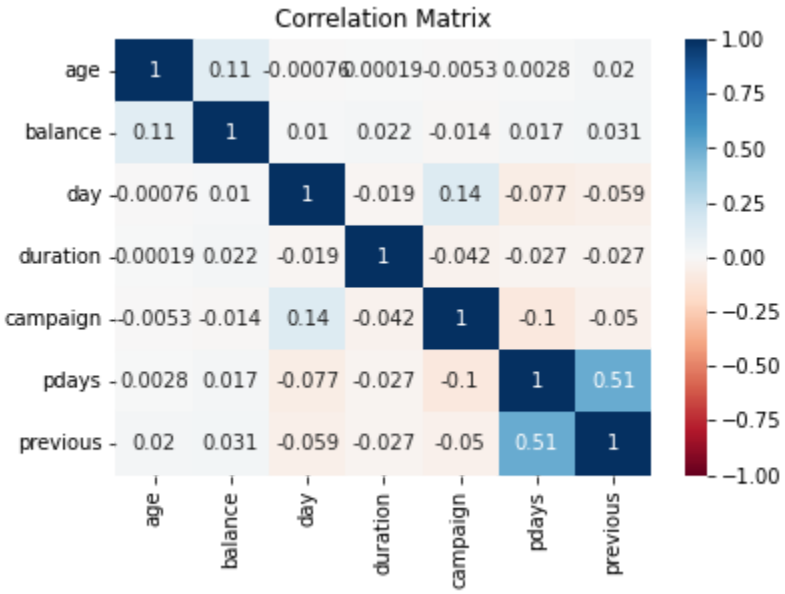
From the column 'previous' we can see that most of the clients have been already contacted, since the value is close to 1 (0.83256)

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        11162 non-null  int64 
 1   job        11162 non-null  object
 2   marital    11162 non-null  object
 3   education  11162 non-null  object
 4   default    11162 non-null  object
 5   balance    11162 non-null  int64 
 6   housing    11162 non-null  object
 7   loan       11162 non-null  object
 8   contact    11162 non-null  object
 9   day        11162 non-null  int64 
 10  month      11162 non-null  object
 11  duration   11162 non-null  int64 
 12  campaign   11162 non-null  int64 
 13  pdays      11162 non-null  int64 
 14  previous   11162 non-null  int64 
 15  poutcome   11162 non-null  object
 16  deposit    11162 non-null  object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

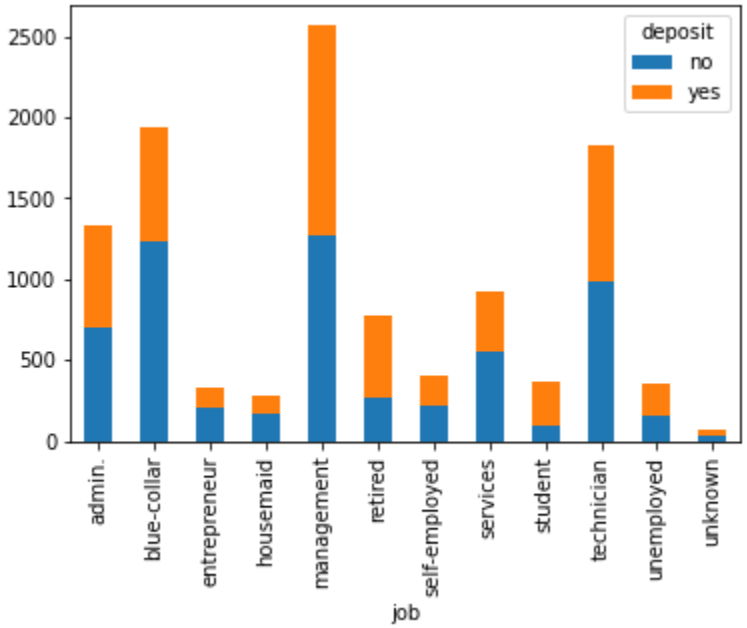
```
#scan the dataset for null or missing values
missing_values = data.isnull().mean()*100

missing_values.sum()
```

The output was 0.0 missing values.

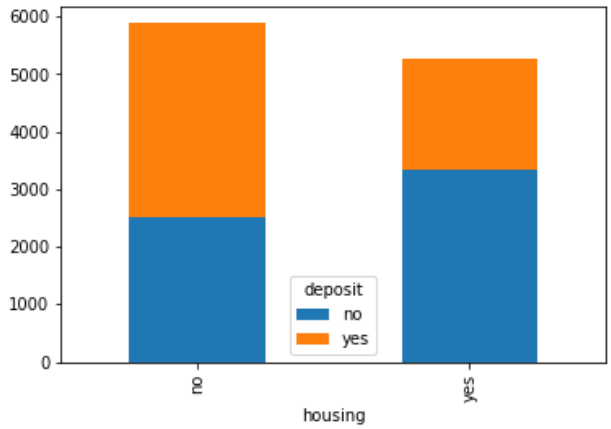


Since our dataset has a combination of categorical and numerical data as columns, we can further explore both to get a better understanding.

```
#Let's explore the categorical columns
categorical_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month','poutcome', 'deposit']


#explore which jobs are likely to result in a deposit
(data
 .groupby(['job', 'deposit'])
 .size()
 .unstack()
 .plot.bar(stacked=True)
)
```

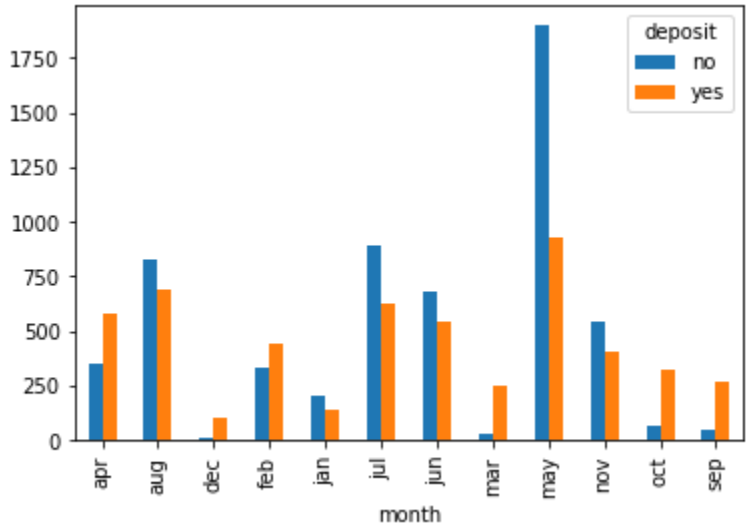



```
#explore how housing influences the chances of a deposit
(data
 .groupby(['housing', 'deposit'])
 .size()
 .unstack()
 .plot.bar(stacked=True)
)
```

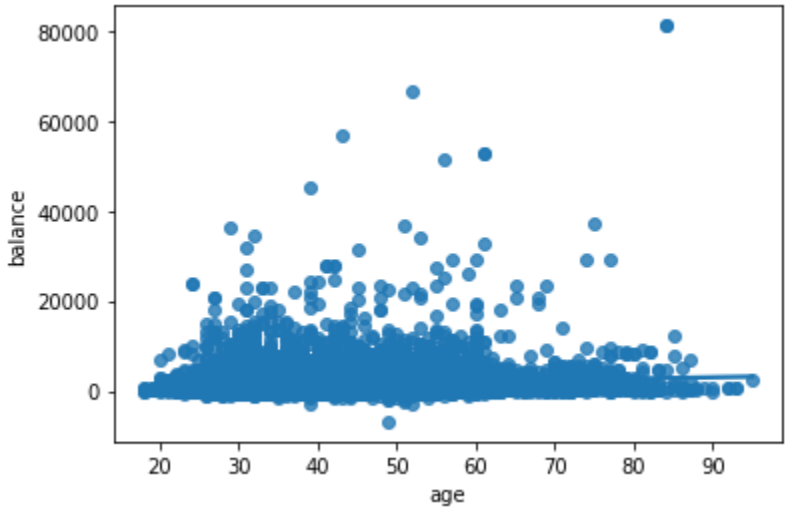



```
#explore best time of the year to reach clients
(data
```

```
 .groupby(['month','deposit'])
 .size()
 .unstack()
 .plot.bar()
)
```

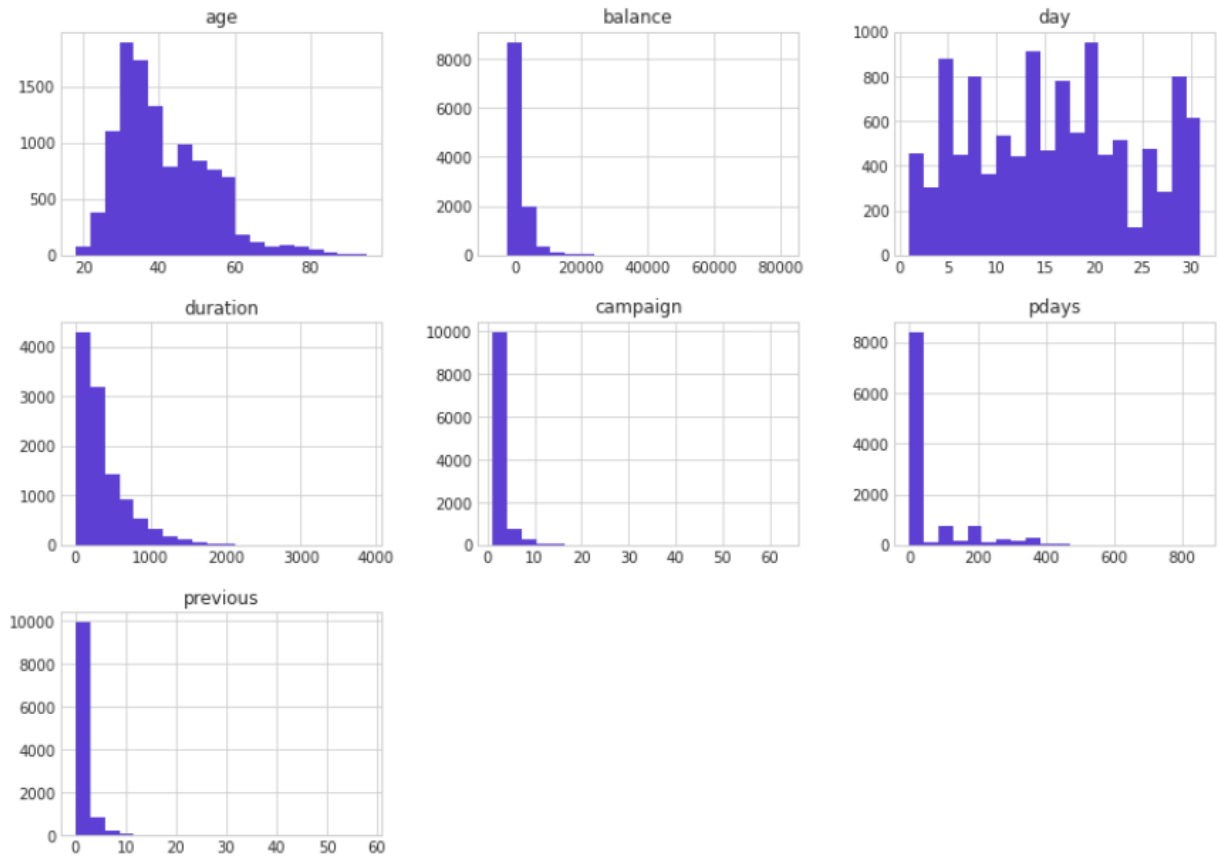



```
#compare age and balance

sns.regplot(x='age',y='balance', data=data)

plt.show()
```



Histogram Grid Comparing Multiple Factors

```
plt.style.use('seaborn-whitegrid')

data.hist(bins=20, figsize=(14,10), color='#5D3FD3')
plt.show()
```



Data preparation for ML algorithms (encoding)

```
#clean data and prepare for machine learning model
#our dataset has categorical data so we need to prepare the dataset for processing

def dummy_bool_values(row, column_name):
    return 1 if row[column_name] == 'yes' else 0

def corrected_values(row, column_name, threshold, data):
    if row[column_name] <= threshold:
        return row[column_name]
    else:
        mean = data[data[column_name] <= threshold][column_name].mean()
        return mean

def processed_df(data):
    cleaned_data = data.copy()

    boolean_columns = ['default', 'housing', 'loan', 'deposit']
    for bool_columns in boolean_columns:
        cleaned_data[bool_columns + '_bool'] = data.apply(lambda row:
dummy_bool_values(row, bool_columns),axis=1)

    cleaned_data = cleaned_data.drop(columns = boolean_columns)

    #now we can convert categorical columns to dummies

    categorical_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

    for new_col in  categorical_columns:
        cleaned_data = pd.concat([cleaned_data.drop(new_col, axis=1),
                          pd.get_dummies(cleaned_data[new_col], prefix=new_col, prefix_sep='_',
                                drop_first=True, dummy_na=False)], axis=1)

    #impute the incorrect values and drop original columns
    cleaned_data['campaign_cleaned'] = data.apply(lambda row: corrected_values(row,
'campaign', 34, cleaned_data),axis=1)
    cleaned_data['previous_cleaned'] = data.apply(lambda row: corrected_values(row,
'previous', 34, cleaned_data),axis=1)

    cleaned_data = cleaned_data.drop(columns = ['campaign', 'previous'])


    return cleaned_data
```

```
#clean the dataset
cleaned_data = processed_df(data)
cleaned_data.head()
```

	age	balance	day	duration	pdays	default_bool	housing_bool	loan_bool	deposit_bool	job_blue-collar	...
0	59	2343	5	1042	-1	0	1	0	1	0	...
1	56	45	5	1467	-1	0	0	0	1	0	...
2	41	1270	5	1389	-1	0	1	0	1	0	...
3	55	2476	5	579	-1	0	1	0	1	0	...
4	54	184	5	673	-1	0	0	0	1	0	...

5 rows × 43 columns

ue-	...	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_other	poutcome_success	poutcome_unknown	campaign_cleaned	previous_cleaned
	...	0	1	0	0	0	0	0	1	1.0	0.0
	...	0	1	0	0	0	0	0	1	1.0	0.0
	...	0	1	0	0	0	0	0	1	1.0	0.0
	...	0	1	0	0	0	0	0	1	1.0	0.0
	...	0	1	0	0	0	0	0	1	2.0	0.0

5 rows × 43 columns

```
#splitting my cleaned data set
X = cleaned_data.drop(columns = 'deposit_bool')
y = cleaned_data[['deposit_bool']]

#setting test size and random state
TEST_SIZE = 0.2
RAND_STATE = 42

#split the dataset and print out the shapes for each set to verify
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = TEST_SIZE,
random_state=RAND_STATE)


display('This is how the dataset is split', X_train.shape, y_train.shape, X_test.shape,
y_test.shape)
```

Output:

```
'This is how the dataset is split'
```

```
(8929, 42)
(8929, 1)
(2233, 42)
(2233, 1)
```

```
#Gaussian Naive Bayes
nb = GaussianNB()


#Random Forest Classifier
rfc = RandomForestClassifier()
```

```
y_train = np.ravel(y_train)

#Gaussian Naive Bayes
nb.fit(X_train,y_train)

#Random Forest Classifier
rfc.fit(X_train,y_train)
```

```
#Gaussian Naive Bayes
print(y_res1)

#Random Forest Classifier
print(y_res2)
```

Output:

```
[1 0 0 ... 0 1 1]
[1 1 1 ... 0 1 1]
```

```
print(y_test)
```

```
      deposit_bool
5527             0
4541             1
1964             1
5007             1
8928             0
...            ...
376              1
5544             0
10749            0
3881             1
6786             0

[2233 rows x 1 columns]
```

Training 2 ML models and comparing accuracy

```
#Gaussian Naive Bayes Accuracy Result
r1 = sklearn.metrics.confusion_matrix(y_test, y_res1)
accuracy1 = (r1[0][0] + r1[-1][-1]) / np.sum(r1)
print(accuracy1)

#Random Forest Classifier Accuracy Result
r2 = sklearn.metrics.confusion_matrix(y_test, y_res2)
accuracy2 = (r2[0][0] + r2[-1][-1]) / np.sum(r2)
print(accuracy2)
```

Output:

```
0.7393640841916704
0.8343036274070756
```

Model	Accuracy (%)
Naive Bayes	73.9%
Random forest	83.4%

Conclusion

Understanding the business problem, evaluating how the dataset is built, and interpreting the categorical/numerical variables at hand is essential for building an accurate and significant model.

We used Gaussian Naive Bayes and Random Decision Forest as models to predict if a client would open a term deposit from the bank.

Gaussian Naive Bayes model has an accuracy of **73.94%**.
Random Decision Classifier has an accuracy of **83.52%**.