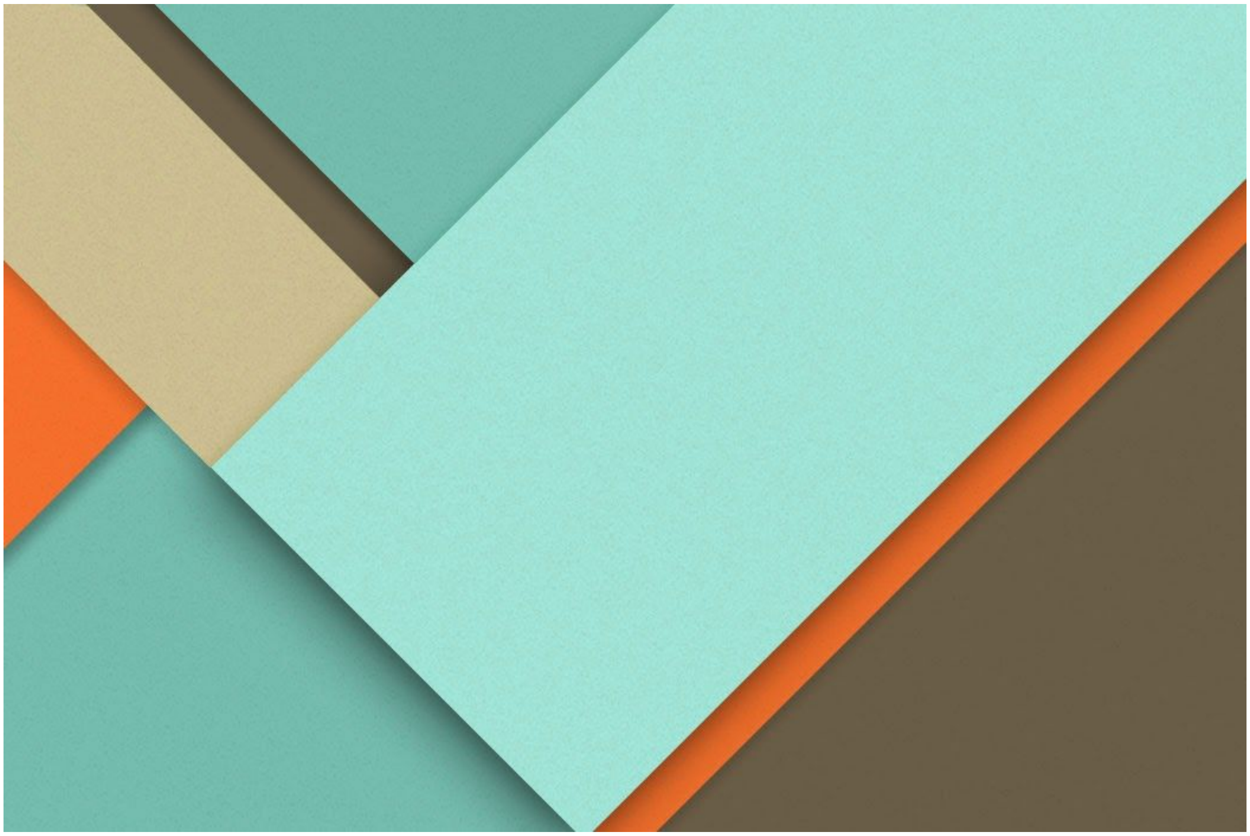


Annual Investment Subscription

(Data Science Semester Project)

18.11.2020

Submitted by:

Raghav Lakhotia (18ucs058)

Submitted To:

Dr. Subrat Dash

Dr. Sakthi Balan

Dr. Sudheer Sharma

INDEX

Project Overview

This Machine Learning Project is based on direct marketing campaigns of a Portuguese banking institution. In order to access if the product (bank term deposit/ Annual Investment) would be (or not) subscribed by the client.

Goals

The classification goal is to predict if the client will subscribe to a term deposit (variable y).

1. Predicting the future results of marketing companies based on available statistics and, accordingly, formulating recommendations for such companies in the future.
2. Building a profile of a consumer of banking services (deposits)

Dataset Specifications

This is a dataset that describes Portugal bank marketing campaign results.

Conducted campaigns were based mostly on direct phone calls, offering bank's clients to place a term deposit.

If after all marketing efforts the client had agreed to place a deposit - the target variable marked 'yes', otherwise 'no'.

Number of Instances:

As the Dataset is huge, we will first apply our algorithm for Small Dataset then scale it up to Big dataset.

- 45,211 for bank-full.csv
- 4,119 for bank-additional.csv

Number of Attributes:

20 Input Attributes

1 Output Attribute.

Dataset Source:

https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Feature Description

Bank Client Data:

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5 - default: has credit in default? (categorical: 'no','yes','unknown')

6 - housing: has housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown')

Related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes the last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

17 - cons.price.idx: consumer price index - monthly indicator (numeric)

18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

Plan of Attack:

1. Importing Libraries
2. Data Collection/ Reading Test Data
3. Data Preparations and feature analysis
4. Data Preprocessing
5. Choice of metrics using ROC
6. Choice of the most effective model by build learning curve rate
7. Train Best model
8. Evaluate the Best Model
9. Make Predictions
10. Conclusions and Recommendations

1. Importing Libraries

```
# Basic Maths and Array libraries
import pandas as pd
import numpy as np
import time
import gc
import warnings
warnings.filterwarnings("ignore")

#Classification Algorithms libraries
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
import category_encoders as ce

#Report Generation
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
from tqdm import tqdm
```

2. Data Collection/ Reading Test Data

Importing Dataset(Small and Big)

As this is a huge dataset, we will apply and train our algorithm on a small dataset, and when it's mature then we will find the result on a big dataset.

Applying the algorithm directly on a huge dataset may emerge a use compilation time.

Here we use Pandas Library to read the CSV File and as in dataset it is separated not by a comma but semicolon so we put the separation criteria as Semicolon(;)

```
# Read Small Dataset then gradually Scal up to Big Data set
data = pd.read_csv('bank-additional.csv', sep=';')
print('There is {} observations with {} features'.format(data.shape[0], data.shape[1]))

# Read Big Dataset
data_full = pd.read_csv('bank-additional-full.csv', sep=';')
print('There is {} Instances with {} features'.format(data_full.shape[0], data_full.shape[1]))
```

```
There is 4119 observations with 21 features
There is 41188 Instances with 21 features
```

Displaying Table

Here are the 10 Rows of Observation to keep a check and visualize the attributes.

```
display(data.head(10))
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	...	2	999	0	nonexistent
1	39	services	single	high.school	no	no	no	telephone	may	fri	...	4	999	0	nonexistent
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	...	1	999	0	nonexistent
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	...	3	999	0	nonexistent
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	...	1	999	0	nonexistent
5	32	services	single	university.degree	no	no	no	cellular	sep	thu	...	3	999	2	failure
6	32	admin.	single	university.degree	no	yes	no	cellular	sep	mon	...	4	999	0	nonexistent
7	41	entrepreneur	married	university.degree	unknown	yes	no	cellular	nov	mon	...	2	999	0	nonexistent
8	31	services	divorced	professional.course	no	no	no	cellular	nov	tue	...	1	999	1	failure
9	35	blue-collar	married	basic.9y	unknown	no	no	telephone	may	thu	...	1	999	0	nonexistent

10 rows × 21 columns

The datatype of attributes imported from the dataset.

Data types of all the attributes before preprocessing

```
In [35]:    data.dtypes

Out[35]: age                 int64
         job                object
         marital            object
         education          object
         default            object
         housing            object
         loan               object
         contact            object
         month              object
         day_of_week        object
         duration            int64
         campaign            int64
         pdays               int64
         previous            int64
         poutcome           object
         emp.var.rate      float64
         cons.price.idx    float64
         cons.conf.idx     float64
         euribor3m         float64
         nr.employed       float64
         y                  object
         dtype: object
```

3. Data Preparations and feature analysis

Now, We will analyze the data, so we have basically two types of data.

1. Categorical Attributes
2. Numeric Attributes.

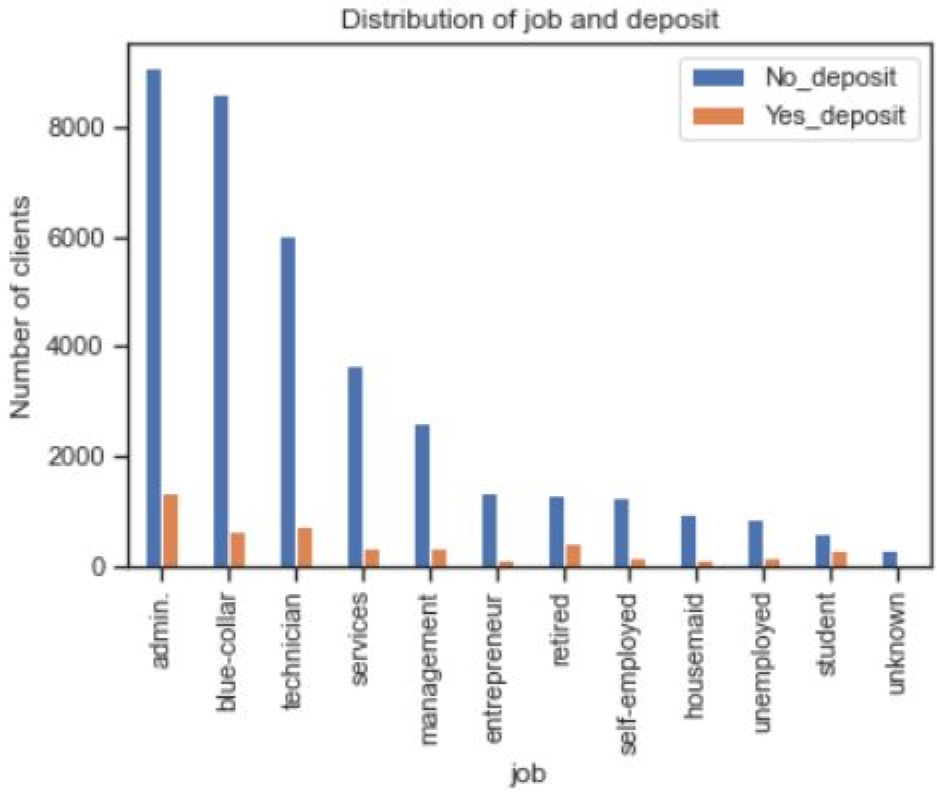
1. Explore categorical features (EDA)

First, we will construct a function that will plot a graph between the categorical attributes and Y. So we will understand how much the impact of this is on the output variable. This will help to prioritize which attribute to consider.

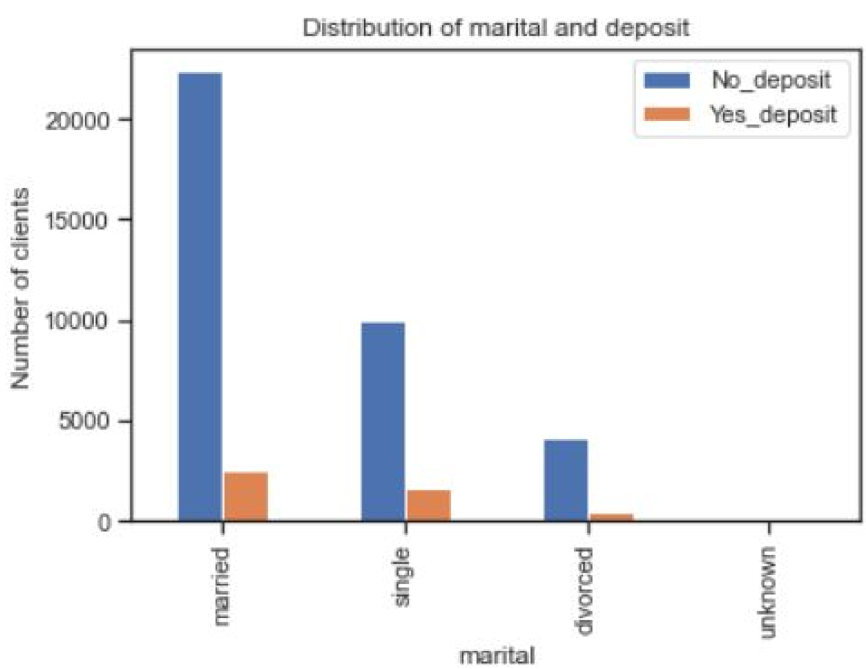
```
# Function to show categorical values disribution
def plot_bar(column):
    # temp df
    temp_1 = pd.DataFrame()

    # count categorical values
    temp_1['No_deposit'] = data[data['y'] == 'no'][column].value_counts()
    temp_1['Yes_deposit'] = data[data['y'] == 'yes'][column].value_counts()
    temp_1.plot(kind='bar')

    plt.xlabel(f'{column}')
    plt.ylabel('Number of clients')
    plt.title('Distribution of {} and deposit'.format(column))
    plt.show();
```

```
# Calling function plot_bar with all categorical Feautures.
plot_bar('job'),
plot_bar('marital'),
plot_bar('education'),
plot_bar('contact'),
plot_bar('loan'),
plot_bar('housing')
```

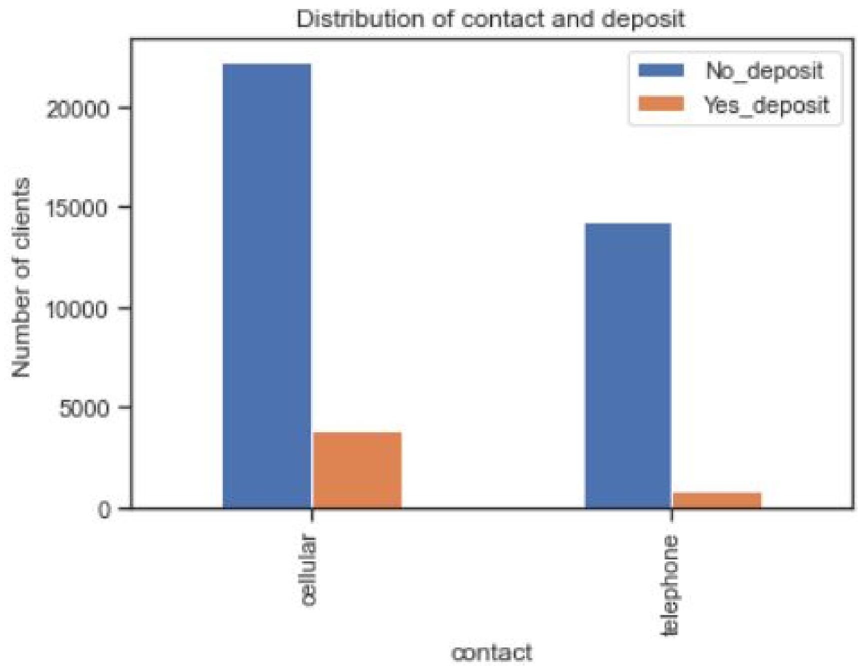
Categorical Feature Job vs Output Variable



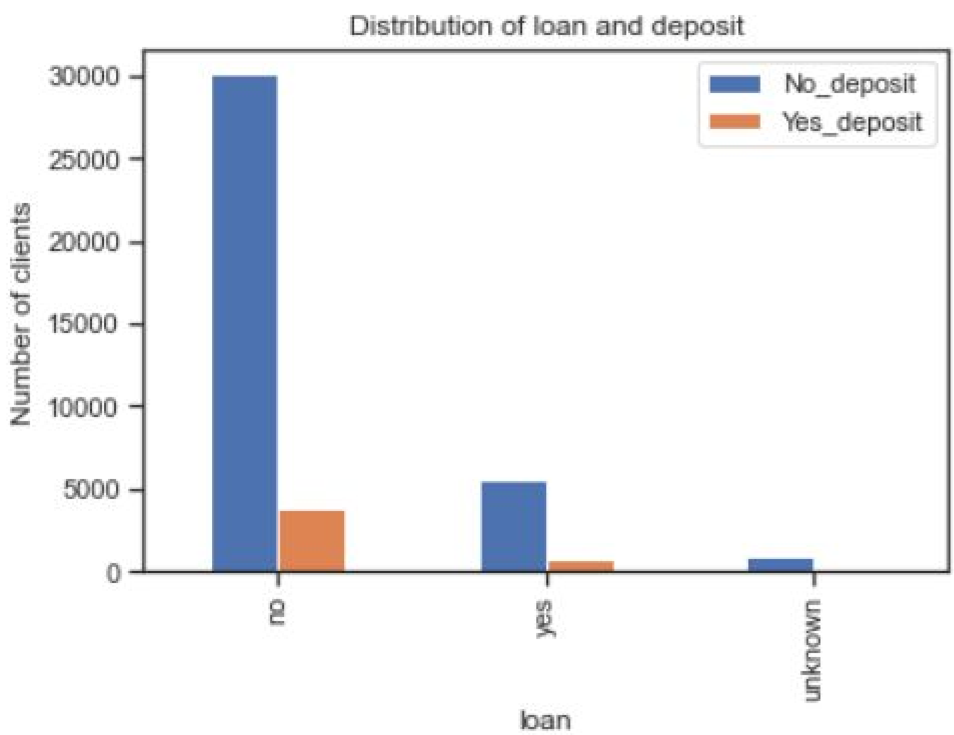
Categorical Feature Marital vs Output Variable



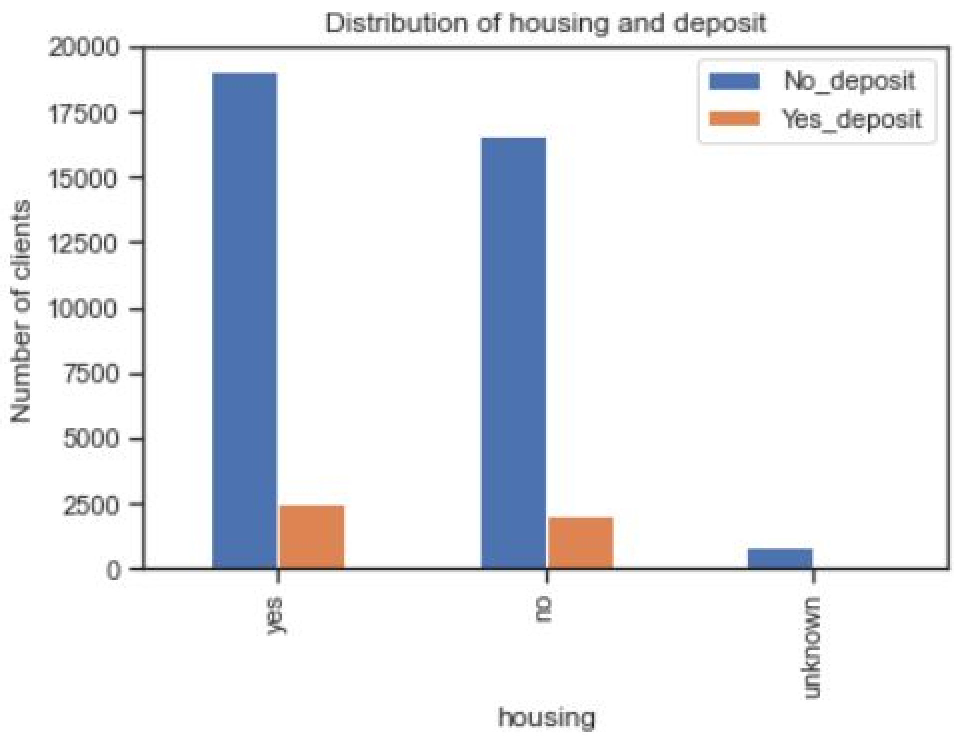
Categorical Feature Contract vs Output Variable



Categorical Feature Loan vs Output Variable



Categorical Feature Housing vs Output Variable



The primary analysis of Several categorical features reveals:

1. **Administrative staff and technical specialists** opened the deposit most of all. In relative terms, a high proportion of pensioners and students might be mentioned as well.

2. Although in absolute terms **married consumers more often** agreed to the service, in relative terms the single responded better.

3. Best communication channel is secular.

4. The difference is evident between consumers who already use the services of banks and received a loan.

5**. Homeownership** does not greatly affect marketing company performance.

2. Explore numerical features (EDA)

Now, We will analyze the numeric Features.

First, we will convert the output variable as binary as yes and no is categorical.

```python
# Convert target variable into numeric
data.y = data.y.map({'no':0, 'yes':1}).astype('uint8')
```

Correlation between features and class for selection

1. Correlation Matrix

```python
# Building  correlation matrix
corr = data.corr()
corr.style.background_gradient(cmap='PuBu')
```

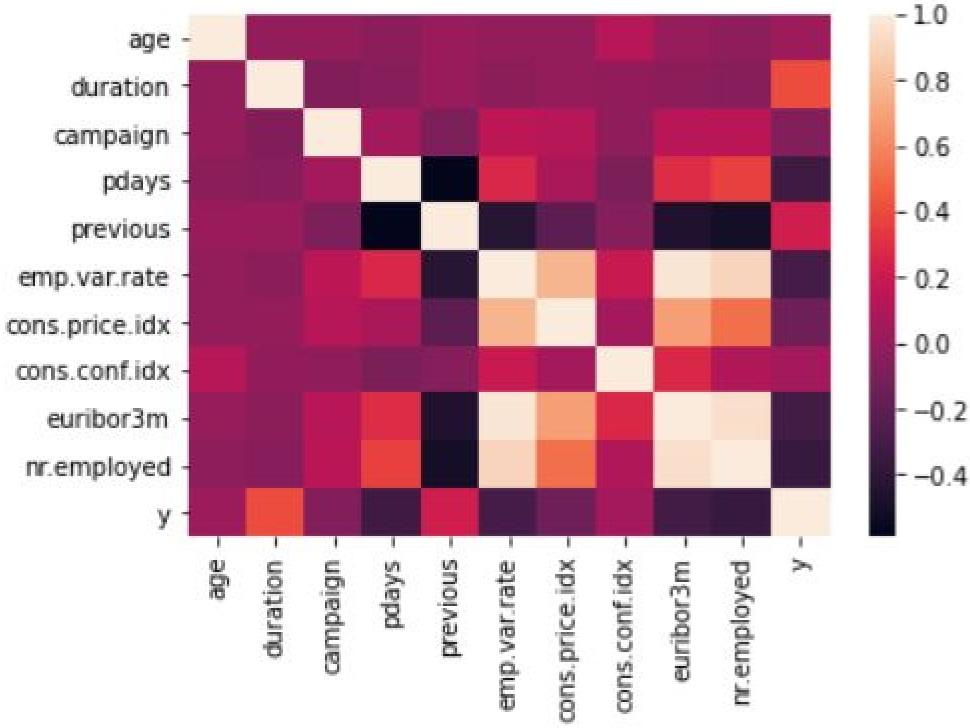
	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
age	1.000000	-0.000866	0.004594	-0.034369	0.024365	-0.000371	0.000857	0.129372	0.010767	-0.017725	0.030399
duration	-0.000866	1.000000	-0.071699	-0.047577	0.020640	-0.027968	0.005312	-0.008173	-0.032897	-0.044703	0.405274
campaign	0.004594	-0.071699	1.000000	0.052584	-0.079141	0.150754	0.127836	-0.013733	0.135133	0.144095	-0.066357
pdays	-0.034369	-0.047577	0.052584	1.000000	-0.587514	0.271004	0.078889	-0.091342	0.296899	0.372605	-0.324914
previous	0.024365	0.020640	-0.079141	-0.587514	1.000000	-0.420489	-0.203130	-0.050936	-0.454494	-0.501333	0.230181
emp.var.rate	-0.000371	-0.027968	0.150754	0.271004	-0.420489	1.000000	0.775334	0.196041	0.972245	0.906970	-0.298334
cons.price.idx	0.000857	0.005312	0.127836	0.078889	-0.203130	0.775334	1.000000	0.058986	0.688230	0.522034	-0.136211
cons.conf.idx	0.129372	-0.008173	-0.013733	-0.091342	-0.050936	0.196041	0.058986	1.000000	0.277686	0.100513	0.054878
euribor3m	0.010767	-0.032897	0.135133	0.296899	-0.454494	0.972245	0.688230	0.277686	1.000000	0.945154	-0.307771
nr.employed	-0.017725	-0.044703	0.144095	0.372605	-0.501333	0.906970	0.522034	0.100513	0.945154	1.000000	-0.354678
y	0.030399	0.405274	-0.066357	-0.324914	0.230181	-0.298334	-0.136211	0.054878	-0.307771	-0.354678	1.000000

Most correlated with the target feature is **Call duration.** So we need to transform it to reduce the influence

Highly correlated features **(employment rate, consumer confidence index, consumer price index)** may describe clients' states from different social-economic angles. Their variance might support the model's capacity for generalization.

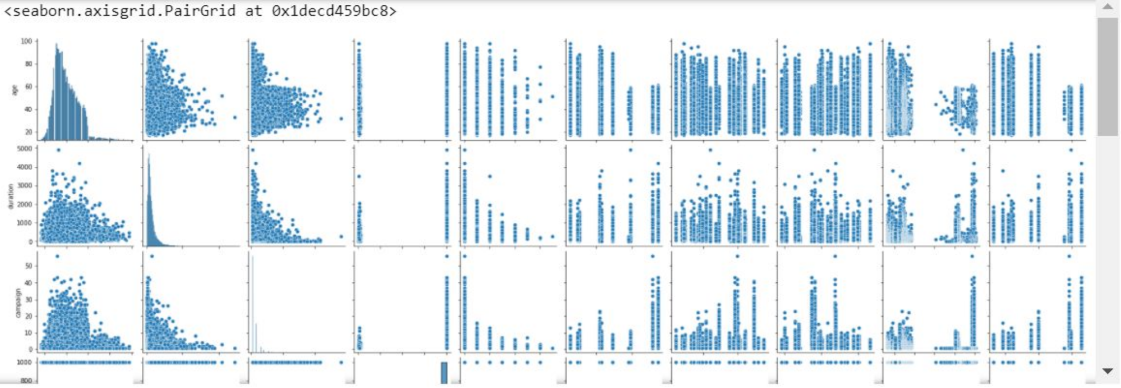
2. Heat plot to visualize the correlation

```
sns.heatmap(corr)
```



3. Pair plot

```
In [3]: sns.pairplot(data)
Out[3]: <seaborn.axisgrid.PairGrid at 0x1decd459bc8>
```



As per the pair plot, correlation matrix, and heatmap, observations as follow:

- Data is non-linear, asymmetric
- Hence selection of features will not depend upon the correlation factor.
- Also, not a single feature is correlated completely with class, hence requires a combination of features.

4. Data Preprocessing

Checking for null values

```
In [8]:  data.isnull().sum()

Out[8]:  age               0
         job               0
         marital           0
         education         0
         default           0
         housing           0
         loan              0
         contact           0
         month             0
         day_of_week       0
         duration          0
         campaign          0
         pdays             0
         previous          0
         poutcome          0
         emp.var.rate      0
         cons.price.idx    0
         cons.conf.idx     0
         euribor3m         0
         nr.employed       0
         y                 0
         dtype: int64
```

Data Clearing and Data preparation for modeling

Since categorical variables dominate in the dataset and the number of weakly correlated numeric variables is not more than 4, we need to transform categorical variables to increase the model's ability to generalize data. (we can not drop them)

Particular attention should be paid to the Duration Feature and categories that can be treated as binary. It suggests using binning and simple transformation accordingly (0 and 1)

For categories of more than 3 types of possible options (marital and education) it is proposed to use the encode targeting - it will allow correctly relation of the values to the target variable and use indicated categories in numerical form.

In some cases, rescaling is proposed to normalize the data

Replacing values with binary ()

```
data.contact = data.contact.map({'cellular': 1, 'telephone': 0}).astype('uint8')
data.loan = data.loan.map({'yes': 1, 'unknown': 0, 'no' : 0}).astype('uint8')
data.housing = data.housing.map({'yes': 1, 'unknown': 0, 'no' : 0}).astype('uint8')
data.default = data.default.map({'no': 1, 'unknown': 0, 'yes': 0}).astype('uint8')
data.pdays = data.pdays.replace(999, 0) # replace with 0 if not contact
data.previous = data.previous.apply(lambda x: 1 if x > 0 else 0).astype('uint8') # binary has contact or not
```

Binary if were was an outcome of a marketing campaign

```
data.poutcome = data.poutcome.map({'nonexistent':0, 'failure':0, 'success':1}).astype('uint8')
```

Change the range of Var Rate

```
data['emp.var.rate'] = data['emp.var.rate'].apply(lambda x: x*-0.0001 if x > 0 else x*1)
data['emp.var.rate'] = data['emp.var.rate'] * -1
data['emp.var.rate'] = data['emp.var.rate'].apply(lambda x: -np.log(x) if x < 1 else np.log(x)).astype('uint8')
```

Multiply consumer index

```
data['cons.price.idx'] = (data['cons.price.idx'] * 10).astype('uint8')
```

Change the sign (we want all be positive values)

```
data['cons.conf.idx'] = data['cons.conf.idx'] * -1
```

Re-scale variables

```
data['nr.employed'] = np.log2(data['nr.employed']).astype('uint8')
data['cons.price.idx'] = np.log2(data['cons.price.idx']).astype('uint8')
data['cons.conf.idx'] = np.log2(data['cons.conf.idx']).astype('uint8')
data.age = np.log(data.age)
```

Less space

```
data.euribor3m = data.euribor3m.astype('uint8')
data.campaign = data.campaign.astype('uint8')
data.pdays = data.pdays.astype('uint8')
```

Function to One Hot Encoding

```
data = encode(data, data.job)
data = encode(data, data.month)
data = encode(data, data.day_of_week)

# Drop tranfromed features
data.drop(['job', 'month', 'day_of_week'], axis=1, inplace=True)
```

Drop the duplicates

```
data.drop_duplicates(inplace=True)
```

Convert Duration Call into 5 categories'

```
def duration(data):
    data.loc[data['duration'] <= 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] <= 180)  , 'duration'] = 2
    data.loc[(data['duration'] > 180) & (data['duration'] <= 319)  , 'duration'] = 3
    data.loc[(data['duration'] > 319) & (data['duration'] <= 645), 'duration'] = 4
    data.loc[data['duration']  > 645, 'duration'] = 5
    return data
duration(data);
```

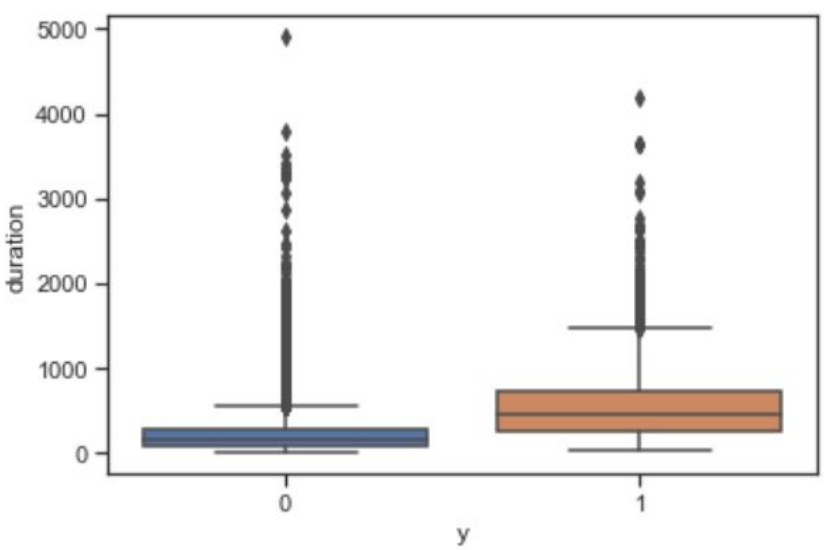
Target encoding for two categorical feature

```
# save target variable before transformation
y = data.y
# Create target encoder object and transoform two value
target_encode = ce.target_encoder.TargetEncoder(cols=['marital', 'education']).fit(data, y)
numeric_dataset = target_encode.transform(data)
# drop target variable
numeric_dataset.drop('y', axis=1, inplace=True)
```

Checking for outliers using boxplots

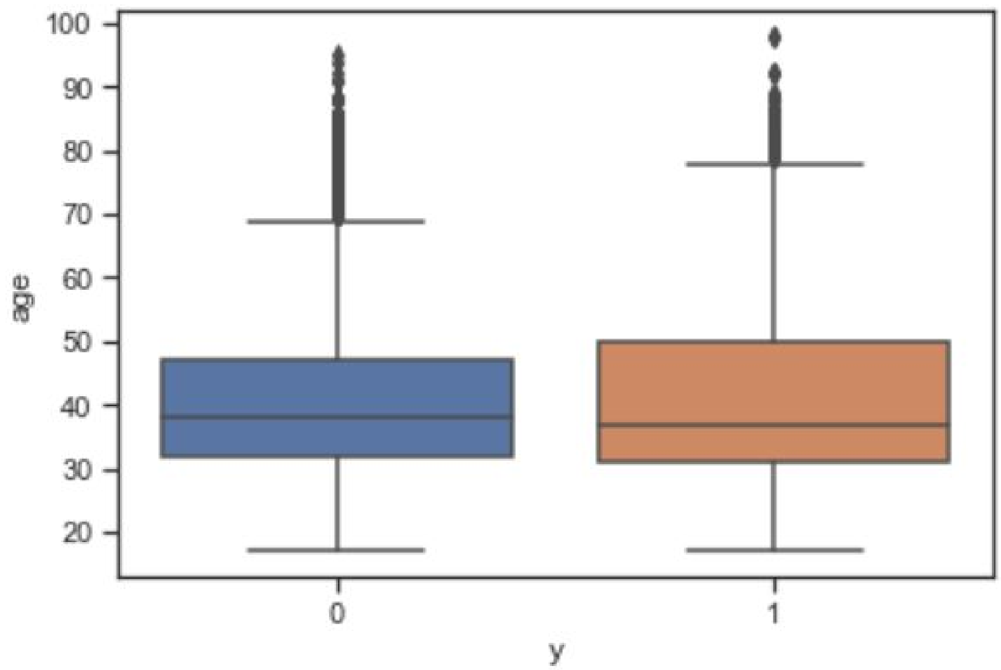
```
sns.boxplot(x='y', y='duration', data=data)
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1f960722828>
```



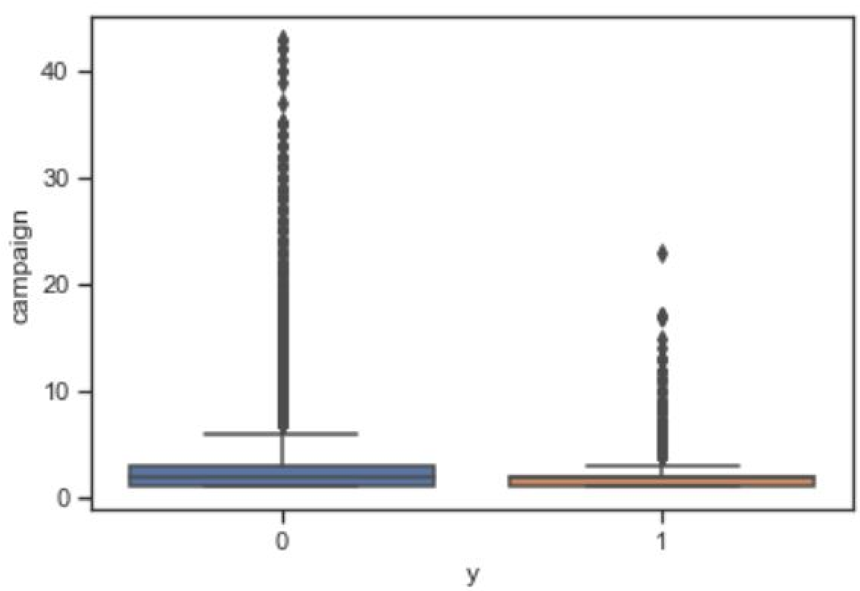
```
sns.boxplot(data['y'],data['age'])
```

23]: <matplotlib.axes._subplots.AxesSubplot at 0x1f960744c88>




```
sns.boxplot(data['y'],data['campaign'])
```

25]: <matplotlib.axes._subplots.AxesSubplot at 0x1f960744c50>



Removing outliers

```python
def remove_outliers(df, col , minimum, maximum):
    col_values = df[col].values
    df[col] = np.where(np.logical_or(col_values<minimum, col_values>maximum), col_values.mean(), col_values)
    return df
```

```python
min_val = data["duration"].min()
max_val = 1500
data = remove_outliers(df=data, column='duration' , minimum=min_val, maximum=max_val)

min_val = data["age"].min()
max_val = 80
data = remove_outliers(df=data, column='age' , minimum=min_val, maximum=max_val)

min_val = data["campaign"].min()
max_val = 6
data = remove_outliers(df=data, column='campaign' , minimum=min_val, maximum=max_val)
```

Data types of all the attributes after preprocessing

```
In [16]:  numeric_dataset.dtypes

Out[16]: age                   float64
         marital               float64
         education             float64
         default                 uint8
         housing                 uint8
         loan                    uint8
         contact                 uint8
         duration                int64
         campaign                uint8
         pdays                   uint8
         previous                uint8
         poutcome                uint8
         emp.var.rate            uint8
         cons.price.idx          uint8
         cons.conf.idx           uint8
         euribor3m               uint8
         nr.employed             uint8
         job_admin.              uint8
         job_blue-collar         uint8
         job_entrepreneur        uint8
         job_housemaid           uint8
         job_management          uint8
         job_retired             uint8
         job_self-employed       uint8
         job_services            uint8
         job_student             uint8
         job_technician          uint8
         job_unemployed          uint8
         job_unknown             uint8
         month_apr               uint8
         month_aug               uint8
         month_dec               uint8
         month_jul               uint8
         month_jun               uint8
         month_mar               uint8
         month_may               uint8
         month_nov               uint8
         month_oct               uint8
         month_sep               uint8
         day_of_week_fri         uint8
         day_of_week_mon         uint8
         day_of_week_thu         uint8
         day_of_week_tue         uint8
         day_of_week_wed         uint8
         dtype: object
```

Splitting the dataset into the Training set and Test set

```
# set global random state
random_state = 11
# split data
X_train, X_test, y_train, y_test = train_test_split(numeric_dataset, y, test_size=0.2, random_state=random_state)
```

```
print('check the shape of splitted train and test sets', X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
check the shape of splitted train and test sets (32940, 44) (32940,) (8235, 44) (8235,)
```

5. Choice of metrics using ROC

ROC (Receiver Operating Characteristic)

- A graphical approach for displaying trade-off between detection rate and false alarm rate
- We use ROC_AUC metrics for evaluating different models with additional monitoring of the accuracy metric dynamic.
- This approach will allow us to explore models from different angles.

Classifiers to Choose Between :

Based on the values of different parameters we can conclude to the following classifiers for Binary Classification.

- Logistics Regression
- Random Forest Classifier
- K Nearest Neighbour
- Decision Tree
- Bagging
- Stochastic Gradient Descent (SGD)

Performance metric using precision and recall calculation along with roc_auc_score & accuracy_score

Building the Pipeline of Classifier for all the Mentioned Algorithm:

```
'''Build pipline of classifiers'''
# set all CPU
n_jobs = -1
# LogisticRegression
pipe_lr = Pipeline([('lr', LogisticRegression(random_state=random_state, n_jobs=n_jobs, max_iter=500))])

# RandomForestClassifier
pipe_rf = Pipeline([('rf', RandomForestClassifier(random_state=random_state, oob_score=True, n_jobs=n_jobs))])

# KNeighborsClassifier
pipe_knn = Pipeline([('knn', KNeighborsClassifier(n_jobs=n_jobs))])

# DecisionTreeClassifier
pipe_dt = Pipeline([('dt', DecisionTreeClassifier(random_state=random_state, max_features='auto'))])

# BaggingClassifier
# note we use SGDClassifier as classier inside BaggingClassifier
pipe_bag = Pipeline([('bag',BaggingClassifier(base_estimator=SGDClassifier(random_state=random_state, n_jobs=n_jobs, max_iter
                                              random_state=random_state,oob_score=True,n_jobs=n_jobs))])
# SGDClassifier
pipe_sgd = Pipeline([('sgd', SGDClassifier(random_state=random_state, n_jobs=n_jobs, max_iter=1500))])
```

```python
'''Set parameters for Grid Search '''
# set number
cv = StratifiedKFold(shuffle=True, n_splits=5, random_state=random_state)
# set for LogisticRegression
grid_params_lr = [{
                'lr__penalty': ['l2'],
                'lr__C': [0.3, 0.6, 0.7],
                'lr__solver': ['sag']
                }]
# set for RandomForestClassifier
grid_params_rf = [{
                'rf__criterion': ['entropy'],
                'rf__min_samples_leaf': [80, 100],
                'rf__max_depth': [25, 27],
                'rf__min_samples_split': [3, 5],
                'rf__n_estimators' : [60, 70]
                }]
# set for KNeighborsClassifier
grid_params_knn = [{'knn__n_neighbors': [16,17,18]}]

# set for DecisionTreeClassifier
grid_params_dt = [{
                'dt__max_depth': [8, 10],
                'dt__min_samples_leaf': [1, 3, 5, 7]
                  }]
# set for BaggingClassifier
grid_params_bag = [{'bag__n_estimators': [10, 15, 20]}]

# set for SGDClassifier
grid_params_sgd = [{
                    'sgd__loss': ['log', 'huber'],
                    'sgd__learning_rate': ['adaptive'],
                    'sgd__eta0': [0.001, 0.01, 0.1],
                    'sgd__penalty': ['l1', 'l2', 'elasticnet'],
                    'sgd__alpha':[0.1, 1, 5, 10]
                    }]
```

```
'''Grid search objects'''
# for LogisticRegression
gs_lr = GridSearchCV(pipe_lr, param_grid=grid_params_lr,
                     scoring='accuracy', cv=cv)
# for RandomForestClassifier
gs_rf = GridSearchCV(pipe_rf, param_grid=grid_params_rf,
                     scoring='accuracy', cv=cv)
# for KNeighborsClassifier
gs_knn = GridSearchCV(pipe_knn, param_grid=grid_params_knn,
                     scoring='accuracy', cv=cv)
# for DecisionTreeClassifier
gs_dt = GridSearchCV(pipe_dt, param_grid=grid_params_dt,
                     scoring='accuracy', cv=cv)
# for BaggingClassifier
gs_bag = GridSearchCV(pipe_bag, param_grid=grid_params_bag,
                     scoring='accuracy', cv=cv)
# for SGDClassifier
gs_sgd = GridSearchCV(pipe_sgd, param_grid=grid_params_sgd,
                     scoring='accuracy', cv=cv)
```

```
# models that we iterate over
look_for = [gs_lr, gs_rf, gs_knn, gs_dt, gs_bag, gs_sgd]
# dict for later use
model_dict = {0:'Logistic_reg', 1:'RandomForest', 2:'Knn', 3:'DesionTree', 4:'Bagging with SGDClassifier', 5:'SGD Class'}
```

```
''' Function to iterate over models and obtain results'''
# set empty dicts and list
result_acc = {}
result_auc = {}
models = []

for index, model in enumerate(look_for):
        start = time.time()
        print()
        print('+++++++ Start New Model ++++++++++++++++++++++')
        print('Estimator is {}'.format(model_dict[index]))
        model.fit(X_train, y_train)
        print('---------------------------------------------')
        print('best params {}'.format(model.best_params_))
        print('best score is {}'.format(model.best_score_))
        auc = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])
        print('---------------------------------------------')
        print('ROC_AUC is {} and accuracy rate is {}'.format(auc, model.score(X_test, y_test)))
        end = time.time()
        print('It lasted for {} sec'.format(round(end - start, 3)))
        print('++++++++ End Model ++++++++++++++++++++++++++')
        print()
        print()
        models.append(model.best_estimator_)
        result_acc[index] = model.best_score_
        result_auc[index] = auc
```

Logistics Regression Summary

```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is Logistic_reg
---------------------------------------------
best params {'lr__C': 0.6, 'lr__penalty': 'l2', 'lr__solver': 'sag'}
best score is 0.9092592592592593
---------------------------------------------
ROC_AUC is 0.9216519675583464 and accuracy rate is 0.905525197328476
It lasted for 113.655 sec
++++++++ End Model +++++++++++++++++++++++++++
```

Random Forest Classifier Summary

```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is RandomForest
---------------------------------------------
best params {'rf__criterion': 'entropy', 'rf__max_depth': 25, 'rf__min_samples_leaf': 80, 'rf__min_samples_split': 3, 'rf__n
_estimators': 70}
best score is 0.9035822707953857
---------------------------------------------
ROC_AUC is 0.926898425815701 and accuracy rate is 0.9038251366120219
It lasted for 74.928 sec
++++++++ End Model +++++++++++++++++++++++++++
```

K Nearest Neighbour Summary

```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is Knn
---------------------------------------------
best params {'knn__n_neighbors': 16}
best score is 0.9050394656952034
---------------------------------------------
ROC_AUC is 0.9019898470991563 and accuracy rate is 0.9024893746205221
It lasted for 40.657 sec
++++++++ End Model +++++++++++++++++++++++++++
```

Decision Tree Summary

```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is DesionTree
---------------------------------------------
best params {'dt__max_depth': 8, 'dt__min_samples_leaf': 3}
best score is 0.9043715846994536
---------------------------------------------
ROC_AUC is 0.8231990313816664 and accuracy rate is 0.8971463266545234
It lasted for 1.613 sec
++++++++ End Model +++++++++++++++++++++++++++
```

Bagging Summary

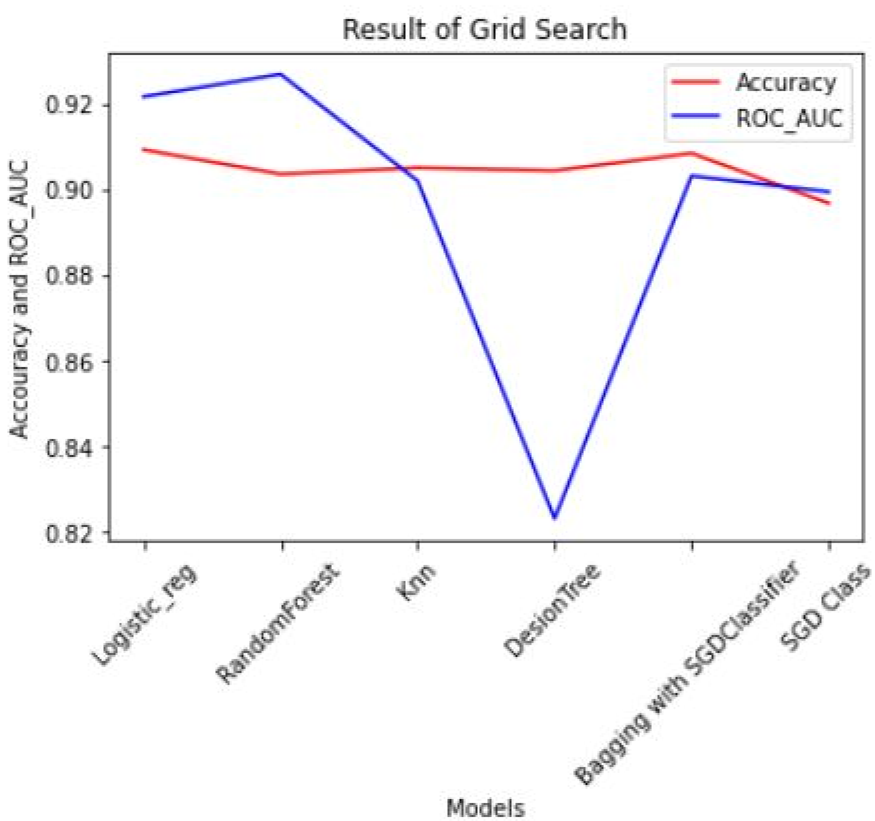
```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is Bagging with SGDClassifier
---------------------------------------------
best params {'bag__n_estimators': 15}
best score is 0.9084395871281117
---------------------------------------------
ROC_AUC is 0.9031566408665629 and accuracy rate is 0.906253794778385
It lasted for 67.871 sec
++++++++ End Model +++++++++++++++++++++++++++
```

Stochastic Gradient Descent (SGD) Summary

```
+++++++ Start New Model ++++++++++++++++++++++
Estimator is SGD Class
---------------------------------------------
best params {'sgd__alpha': 0.1, 'sgd__eta0': 0.001, 'sgd__learning_rate': 'adaptive', 'sgd__loss': 'log', 'sgd__penalty': 'l
2'}
best score is 0.8967820279295691
---------------------------------------------
ROC_AUC is 0.8994453391525427 and accuracy rate is 0.8964177292046145
It lasted for 163.488 sec
++++++++ End Model +++++++++++++++++++++++++++
```

6. Analysis of the most effective model by building curve rate

```
plt.plot(model_dict.values(), result_acc.values(), c='r')
plt.plot(model_dict.values(), result_auc.values(), c='b')
plt.xlabel('Models')
plt.xticks(rotation=45)
plt.ylabel('Accouracy and ROC_AUC')
plt.title('Result of Grid Search')
plt.legend(['Accuracy', 'ROC_AUC'])
plt.show();
```



```
""" Model performance during Grid Search """
pd.DataFrame(list(zip(model_dict.values(), result_acc.values(), result_auc.values())), \
                         columns=['Model', 'Accuracy_rate','Roc_auc_rate'])
```

]:

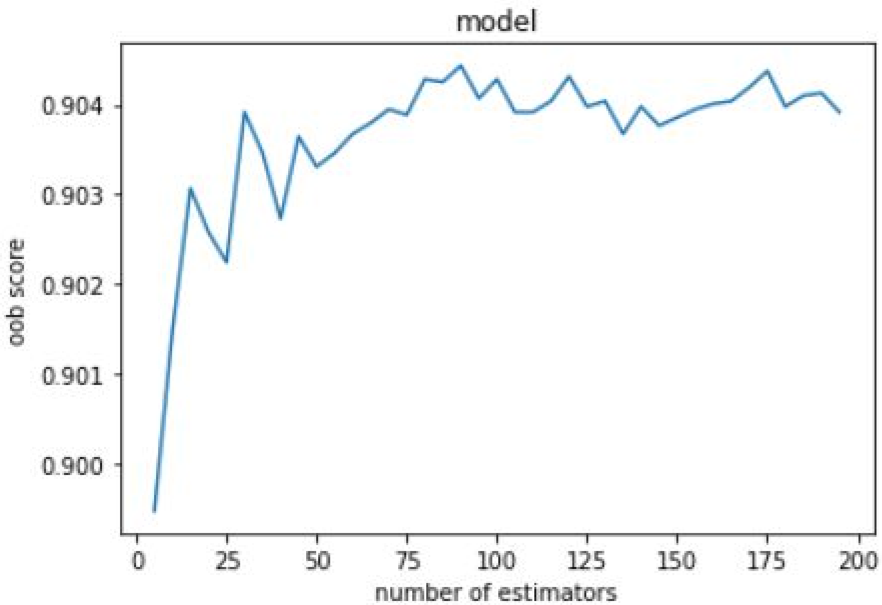
	Model	Accuracy_rate	Roc_auc_rate
0	Logistic_reg	0.909259	0.921652
1	RandomForest	0.903582	0.926898
2	Knn	0.905039	0.901990
3	DesionTree	0.904372	0.823199
4	Bagging with SGDClassifier	0.908440	0.903157
5	SGD Class	0.896782	0.899445

7. Train and Evaluate Best model

Our best performed model with ROC_AUC (0.9269) metric is **Random forest**. This classifier could achieve accuracy rate 0.903 that is average accuracy among all classifiers (0.904).
We can build a graph to check RandomForestClassifier performance with OOB score to be sure that critical hyperparameter was correctly selected during Grid Search. As you may see it almost the same - 80 estimators with best ROC_AUC score and 90 estimators with maximum of OOB score

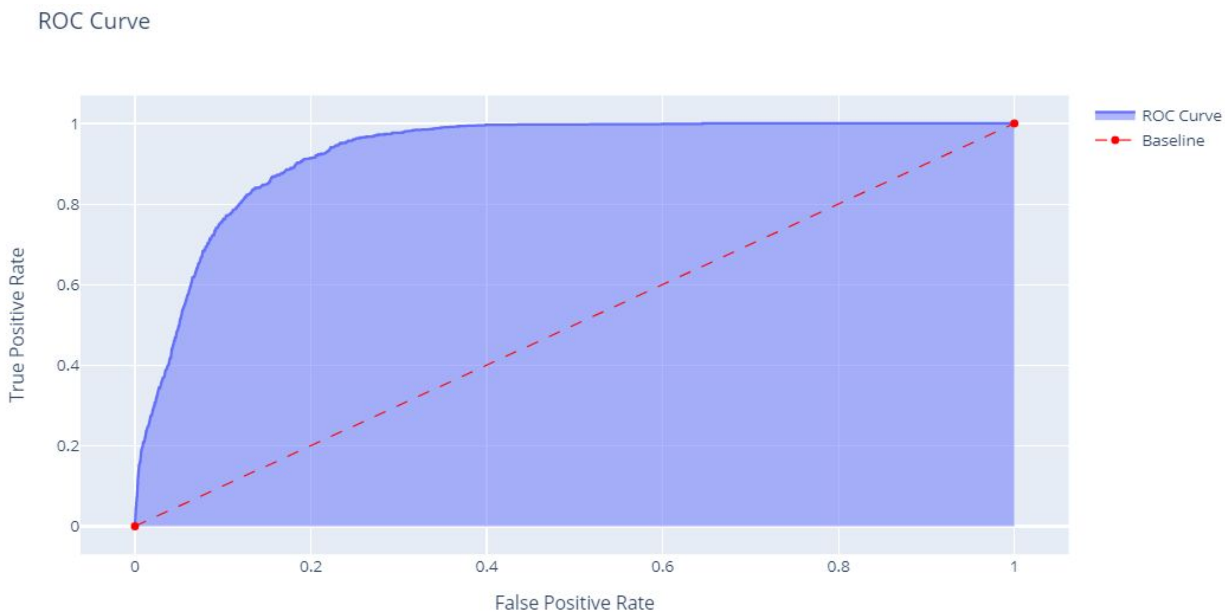
```
def graph(model, X_train, y_train):
    obb = []
    est = list(range(5, 200, 5))
    for i in tqdm(est):
        random_forest = model(n_estimators=i, criterion='entropy', random_state=11, oob_score=True, n_jobs=-1, \
                                 max_depth=25, min_samples_leaf=80, min_samples_split=3,)
        random_forest.fit(X_train, y_train)
        obb.append(random_forest.oob_score_)
    display('max oob {} and number of estimators {}'.format(max(obb), est[np.argmax(obb)]))
    plt.plot(est, obb)
    plt.title('model')
    plt.xlabel('number of estimators')
    plt.ylabel('oob score')
    plt.show();

graph(RandomForestClassifier, X_train, y_train)
```

```
100%|██████████████████████████████████████████████████████████████████| 39/39 [00:51<00:00,  1.32s/it]
```



ROC Curve

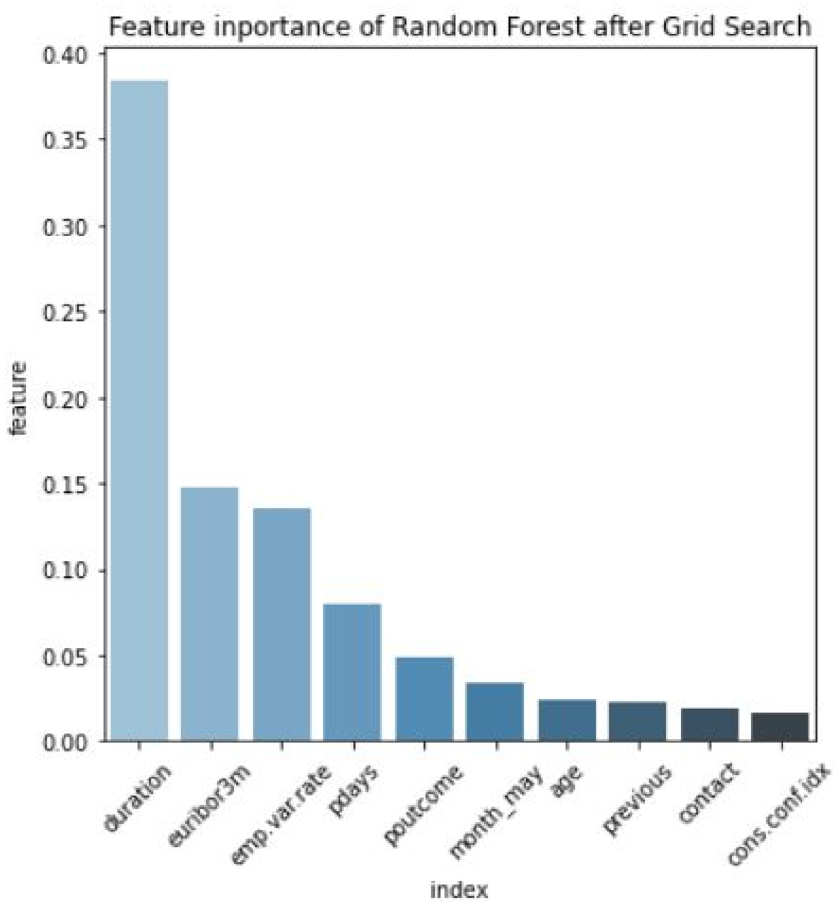
```
''' Build graph for ROC_AUC '''

fpr, tpr, threshold = roc_curve(y_test, models[1].predict_proba(X_test)[:,1])

trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    text=threshold,
    fill='tozeroy',
    name='ROC Curve')

trace1 = go.Scatter(
    x=[0,1],
    y=[0,1],
    line={'color': 'red', 'width': 1, 'dash': 'dash'},
    name='Baseline')

data = [trace0, trace1]

layout = go.Layout(
    title='ROC Curve',
    xaxis={'title': 'False Positive Rate'},
    yaxis={'title': 'True Positive Rate'})

fig = go.Figure(data, layout)
fig.show();
```



Curve is well distributed with tendency to False Positive Rate. The roc auc values of the best model of 0.9269 is quite high level to make later assumptions about the data.

We can build feature importance of RandomForestClassifier with best ROC_AUC score

```python
''' Build bar plot of feature importance of the best model '''

def build_feature_importance(model, X_train, y_train):

    models = RandomForestClassifier(criterion='entropy', random_state=11, oob_score=True, n_jobs=-1, \
                             max_depth=25, min_samples_leaf=80, min_samples_split=3, n_estimators=70)
    models.fit(X_train, y_train)
    data = pd.DataFrame(models.feature_importances_, X_train.columns, columns=["feature"])
    data = data.sort_values(by='feature', ascending=False).reset_index()
    plt.figure(figsize=[6,6])
    sns.barplot(x='index', y='feature', data=data[:10], palette="Blues_d")
    plt.title('Feature inportance of Random Forest after Grid Search')
    plt.xticks(rotation=45)
    plt.show();

build_feature_importance(RandomForestClassifier, X_train, y_train)
```



8. Conclusions, Results and Recommendations

```
# Report Generation
models = RandomForestClassifier(criterion='entropy', random_state=11, oob_score=True, n_jobs=-1, \
                                max_depth=25, min_samples_leaf=80, min_samples_split=3, n_estimators=70)
models.fit(X_train, y_train)

predictions = models.predict(X_test)

print("Accuracy : ", accuracy_score(y_test, predictions))
print("Confusion Matrix : \n",confusion_matrix(y_test, predictions))
print("Classification Report: \n",classification_report(y_test, predictions))
```

```
Accuracy :  0.9038251366120219
Confusion Matrix :
 [[7253   69]
 [ 723  190]]
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.99      0.95      7322
           1       0.73      0.21      0.32       913

    accuracy                           0.90      8235
   macro avg       0.82      0.60      0.64      8235
weighted avg       0.89      0.90      0.88      8235
```

This analysis can be carried out at the level of individual bank branches as it does not require much resources and special knowledge (the model itself can be launched automatically with a certain periodicity).

Potentially similar micro-targeting will increase the overall effectiveness of the entire marketing company.

1. Take into account the time of the company (May is the most effective)
2. Increase the time of contact with customers (perhaps in a different way formulating the goal of the company). It is possible to use other means of communication.
3. Focus on specific categories. The model shows that students and senior citizens respond better to proposal.

4. It is imperative to form target groups based on socio-economic categories. Age, income level (not always high), profession can accurately determine the marketing profile of a potential client.

Given these factors, it is recommended to **concentrate on those consumer groups** that are potentially more promising.

The concentration of the bank's efforts will effectively distribute the company's resources to the main factor - the bank's contact time with the client - it affects most of all on conversion.

The continuation of such a study may be the **formation of a clear customer profile** - by age, gender, income and other factors, as well as the adaptation of the product itself (deposit) for a specific category of consumer.

Reference and Bibliography

https://www.datacamp.com/community/tutorials/kaggle-machine-learning-eda

https://cloud.google.com/blog/products/ai-machine-learning/building-ml-models-with-eda-feature-selection

https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

https://statisticsbyjim.com/basics/correlations/

https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

https://datascience.foundation/sciencewhitepaper/knowing-all-about-outliers-in-machine-learning

https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

https://www.javatpoint.com/machine-learning-random-forest-algorithm