Data Science Internship

Week 13: Data Science Project: Bank Marketing (Campaign)

Final Project Report

Group Name: One
Name: Hira Fahim
Email: hirashahidd26@yahoo.com
Country: United Kingdom
Company: Unemployed
Specialization: Data Science
Batch Code: LISUM19
Submission Date: 12th May 2023
Submitted to: Data Glacier

Table of Contents

1. Problem Description

ABC Bank wants to sell its term deposit product to customers and before launching the product they want to develop a model which help them in understanding whether a particular customer will buy their product or not (based on customer's past interaction with bank or other Financial Institution).

2. Business understanding

2.1. Objectives

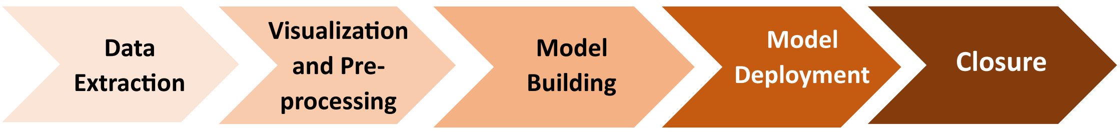
The goal is to build a binary classification model to predict whether the client will subscribe a term deposit or not.

2.2. Strategy

The analysis consists of four parts:

- Data Understanding.
- Perform exploratory analysis.
- Data Visualisation and Pre-processing.
- Model building.
- Model deployment.

3. Project lifecycle



4. Dataset Information

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

5. Attribute Information

5.1. Input Variables

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: 'primary', 'secondary', 'tertiary', 'unknown')
5. default: has credit in default? (Categorical: 'no', 'yes')
6. balance: average yearly balance, in euros (numerical)
7. housing: has housing loan? (Categorical: 'no', 'yes')
8. loan: has personal loan? (Categorical: 'no', 'yes')
   # related with the last contact of the current campaign:
9. contact: contact communication type (categorical: 'cellular', 'telephone', Unknown)

10. day: last contact day of the month (numeric)
11. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
12. duration: last contact duration, in seconds (numerical)
13. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
14. 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)
15. previous: number of contacts performed before this campaign and for this client (numeric)
16. poutcome: outcome of the previous marketing campaign (categorical: 'failure', 'unknown', 'success', 'other')

5.2. Output variable (desired target)

17. y - has the client subscribed a term deposit? (Binary: 'yes', 'no')

5.3. Application Workflow

Given Workflow shows Gradient Boosting Classifier model is used and Flask Framework for deployment. It represents the details of how the model works from user interface till the results.

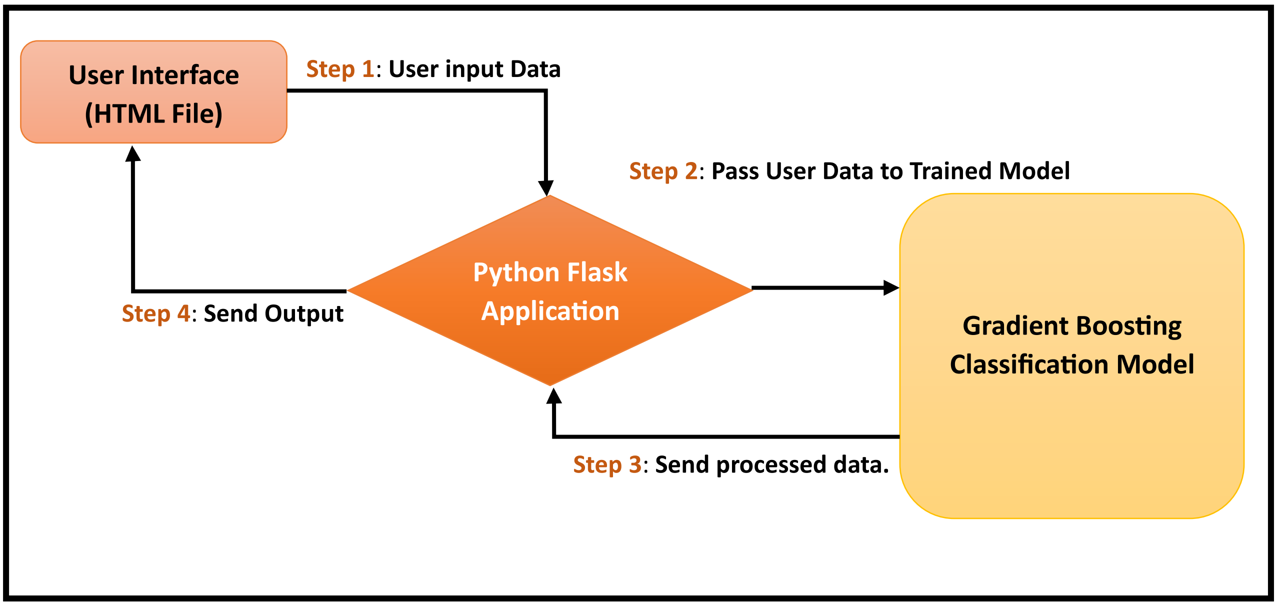


Fig 4.1 Application Framework

The machine learning model is built for Prediction of term deposit subscription based on input attributes, then creates an API for the model using flask Framework and python micro-framework for building web application. This API call used to predict results through HTTP requests.

6. Building Machine Learning Model

6.1. Import Dataset

Import dataset for model training and building.

```
In [1]: import pandas as pd
        import numpy as np

In [2]: d= pd.read_csv("bank-full.csv")

In [3]: d.head()
Out[3]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

6.2. Dataset Details

- Shape of the dataset (Number of rows and columns)

```
In [18]: # no of rows and columns
         d.shape

Out[18]: (45211, 17)
```

Number of rows = 45211
Number of columns = 17

- Datatype of Columns and Non-null values

```
In [19]: # Datatypes of columns and non-null values
         d.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

- Numerical and categorical Features

```
Numeric Features:
Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous'], dtype='object')
===============================================================================
Categorical Features:
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'poutcome', 'y'],
      dtype='object')
```

➢ Null values

```
# total null values in the dataset
d.isnull().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

There are no null values in the dataset.

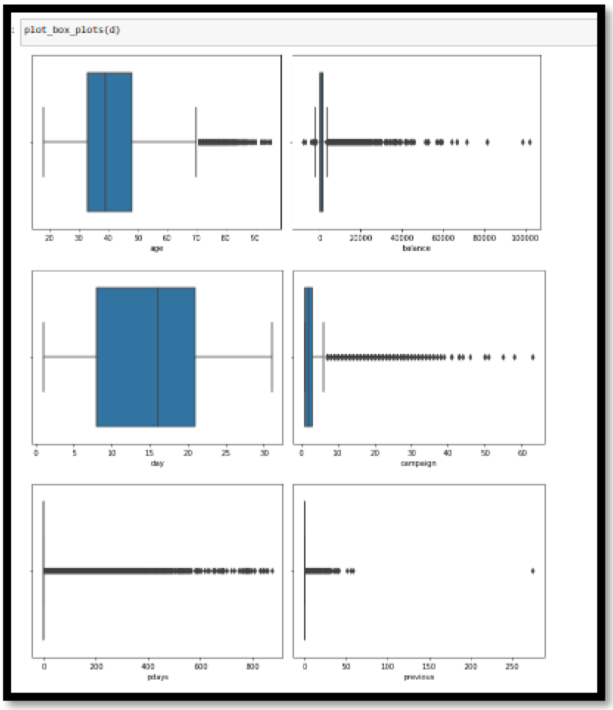
7. Dataset Description and Visualization

➢ Description of Numerical Variables

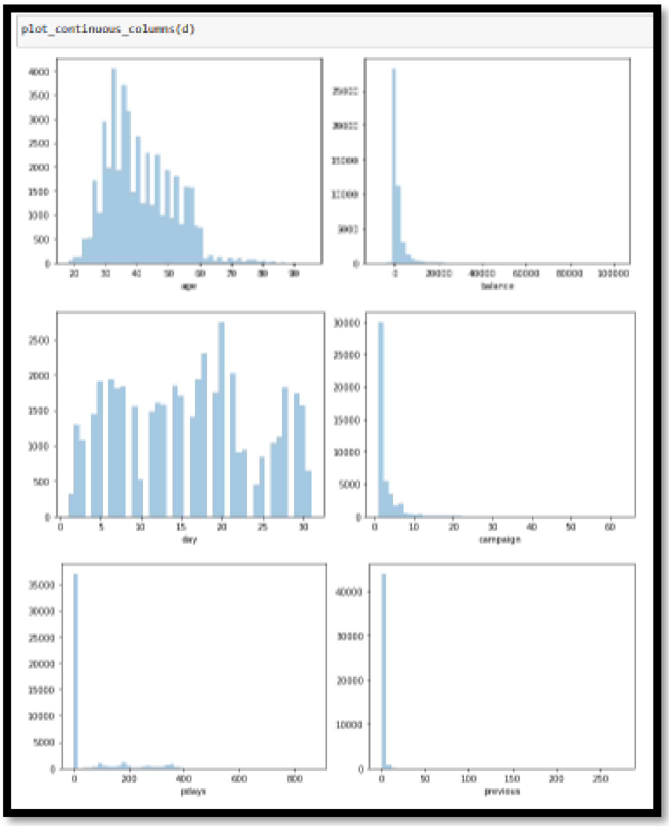
```
# Description of numerical columns
d.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

➢ Numerical variables' visualization

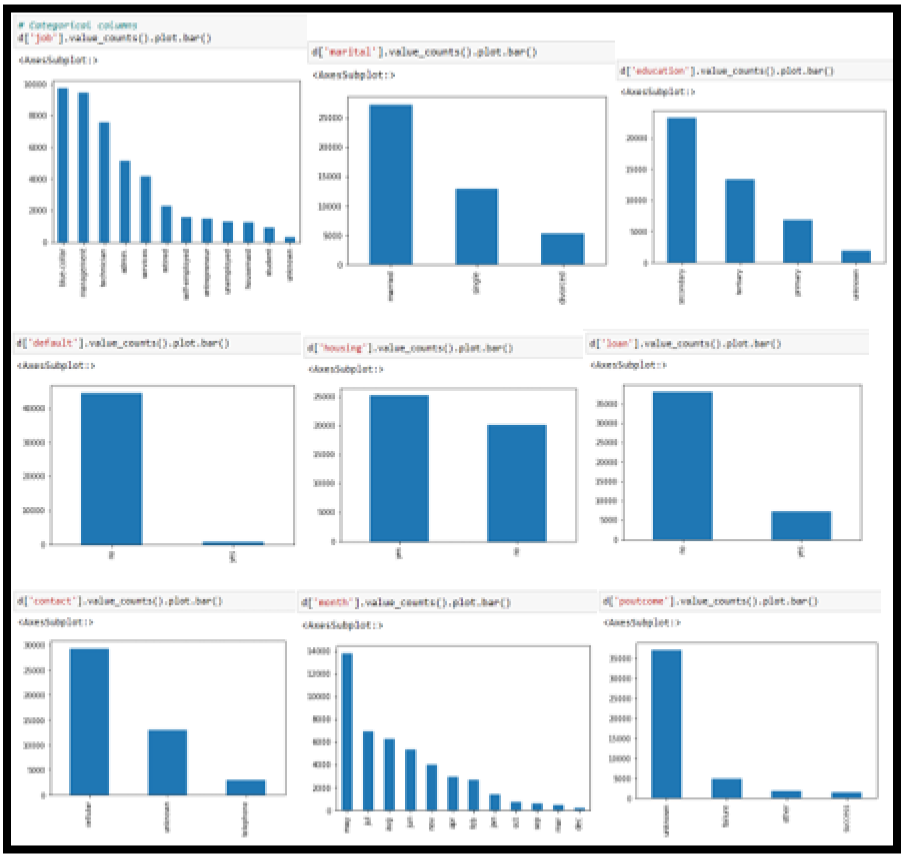


From **description** and **boxplot**, we can see there are outliers in numerical input variables like age, balance, campaign, pdays and previous.



In Histogram, we can see input variables like age, balance, campaign, pdays and previous are **positively skewed**, and we can also see uneven distribution of data in day column.

➤ Categorical data visualization



In **Bar chart** of categorical columns, we see uneven distribution of data in almost all the input categorical columns.

8. Correlation Analysis

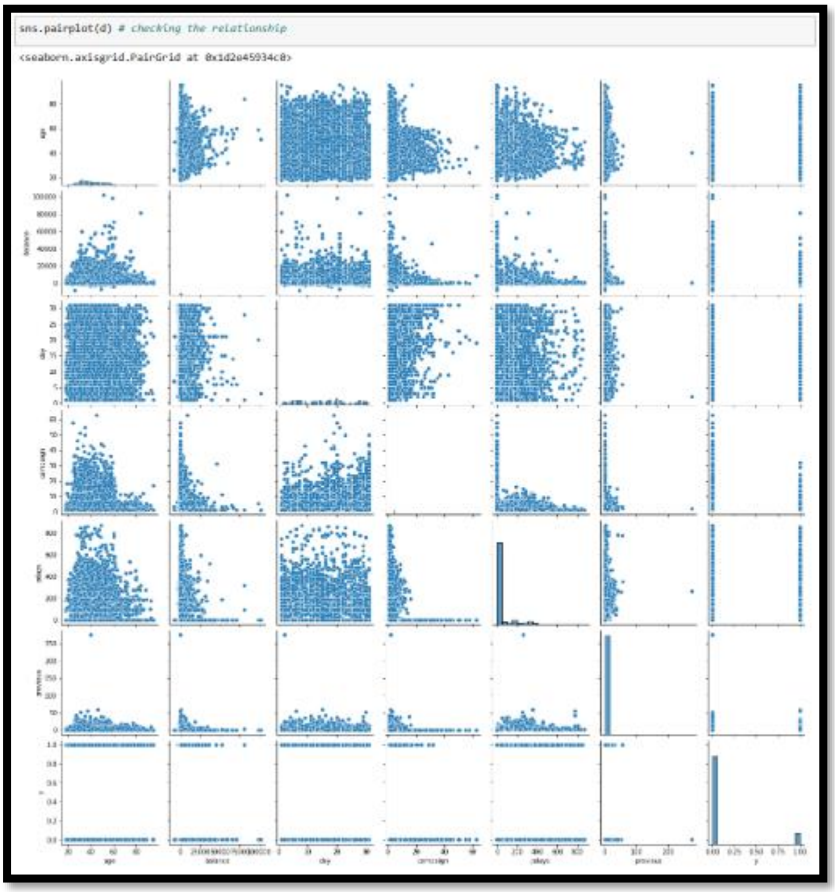
Correlation analysis (or bivariate analysis) examines the relationship between two attributes, say X and Y, and determines whether the two are correlated.

8.1. Correlation between Output and numerical input variables

```
d.corr()
```

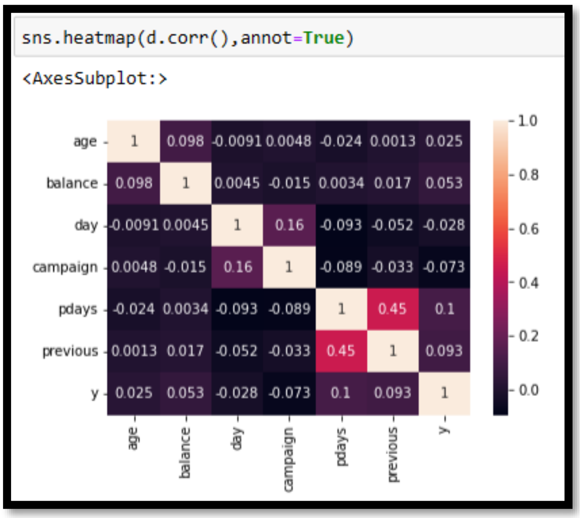
	age	balance	day	campaign	pdays	previous	y
age	1.000000	0.097783	-0.009120	0.004760	-0.023758	0.001288	0.025155
balance	0.097783	1.000000	0.004503	-0.014578	0.003435	0.016674	0.052838
day	-0.009120	0.004503	1.000000	0.162490	-0.093044	-0.051710	-0.028348
campaign	0.004760	-0.014578	0.162490	1.000000	-0.088628	-0.032855	-0.073172
pdays	-0.023758	0.003435	-0.093044	-0.088628	1.000000	0.454820	0.103621
previous	0.001288	0.016674	-0.051710	-0.032855	0.454820	1.000000	0.093236
y	0.025155	0.052838	-0.028348	-0.073172	0.103621	0.093236	1.000000

8.2. Pairplot



As per the **correlation coefficient** and **pairplot**, there is no strong correlation between numerical input variables and output variable.

8.3. Heatmap



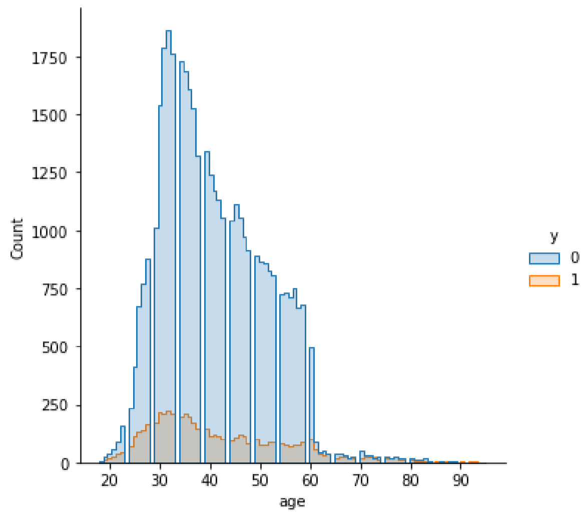
Here we see less correlation between numerical input variable and output variable.

9. Bivariate Analysis

- Term deposit based on Age.

```
sns.displot(d, x="age", hue="y", element="step")
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f0f0cfd0>
```

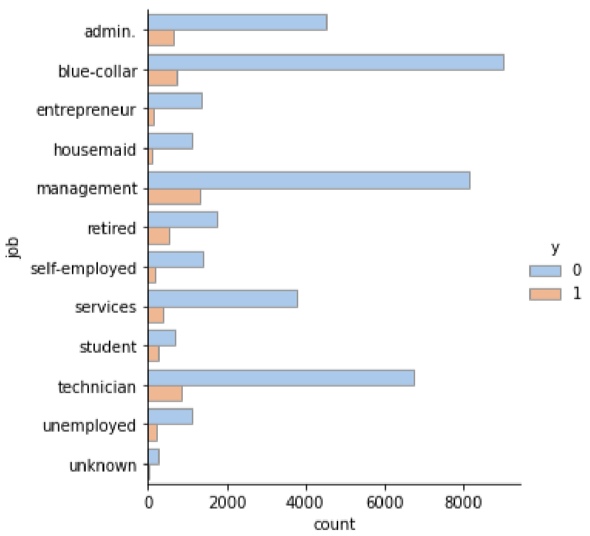


Here we see people between age 30-40 are more responsive towards term deposit.

- Term deposit based on Job Type

```
sns.catplot(
    data=d, y="job", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f108e670>
```

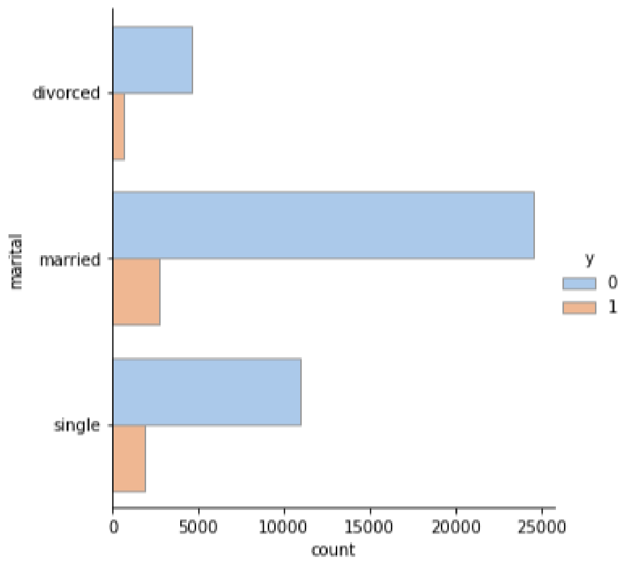


Here we see people with job related to 'management, blue-collar and technician' have subscribed for deposit.

➢ Term deposit based on Marital Status

```
sns.catplot(
    data=d, y="marital", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f0edd9a0>
```

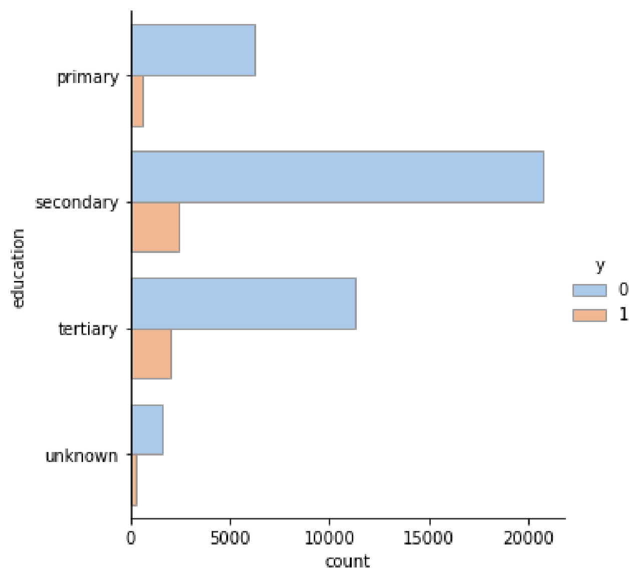


Married people are main contributor for deposit scheme.

➢ Term deposit based on Education Level

```
sns.catplot(
    data=d, y="education", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f121d250>
```

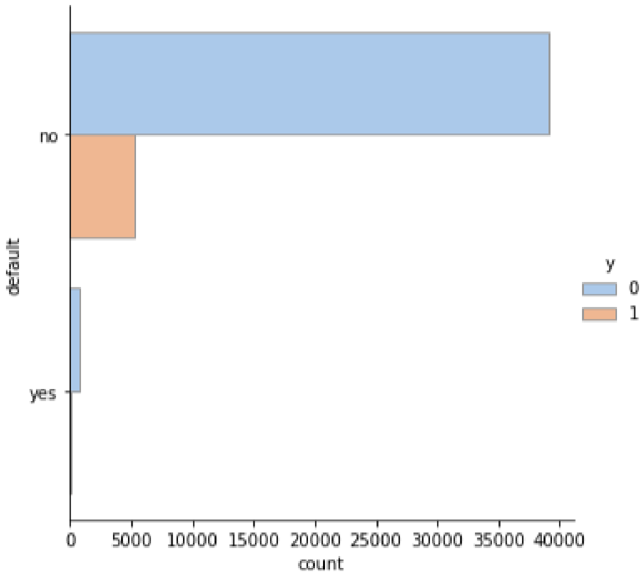


People with secondary and tertiary educational background are main contributors.

➢ Term deposit based on Credit Default

```
sns.catplot(
    data=d, y="default", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f1087ee0>
```

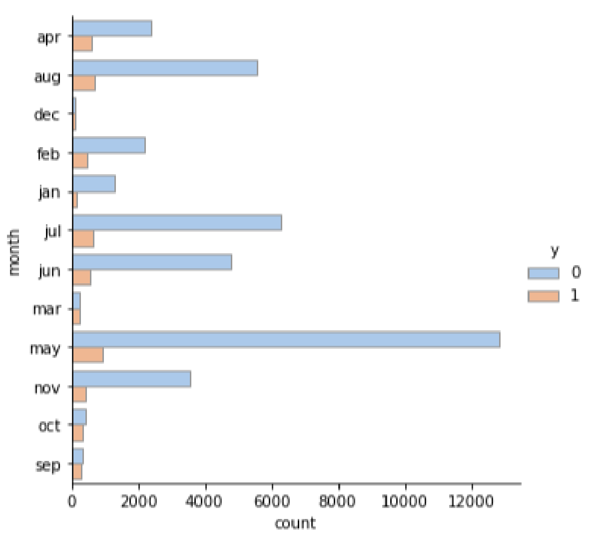


People with good credit history are more interested in term deposit.

➢ Term deposit based on Month.

```
sns.catplot(
    data=d, y="month", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1c2e3214280>
```

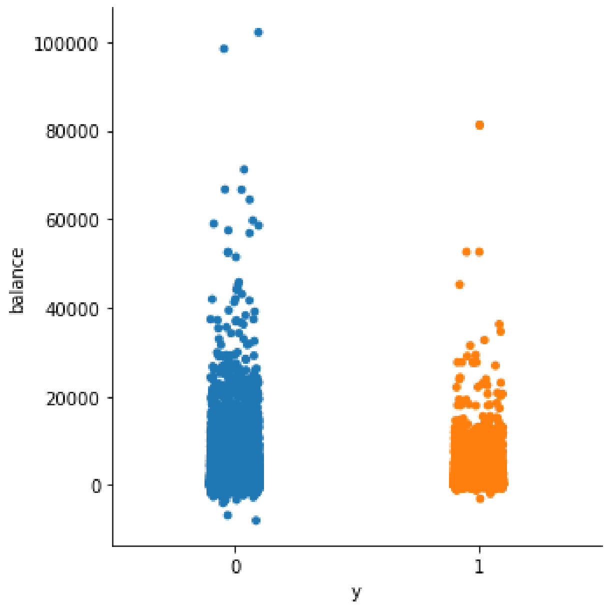


Investment in term deposit is fluctuating throughout the year but in the month of 'May' we have highest success rate.

➢ Term deposit based on Balance.

```
sns.catplot(data=d, x="y", y="balance")
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f1a36af0>
```

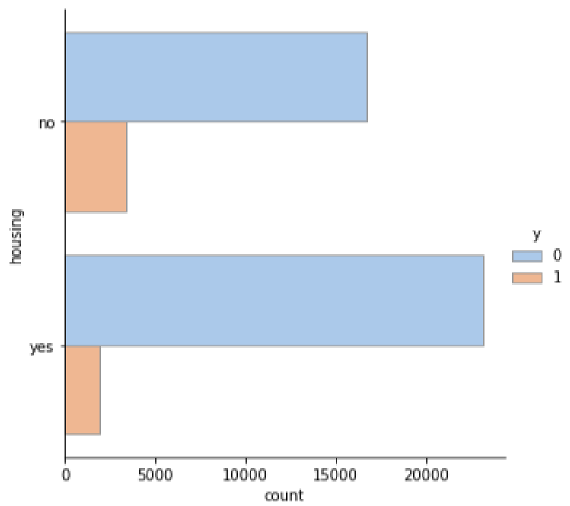


People with balance up to 20,000 are more interested in term deposit.

➢ Term deposit based on Housing Loan

```
sns.catplot(
    data=d, y="housing", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f1c55f70>
```

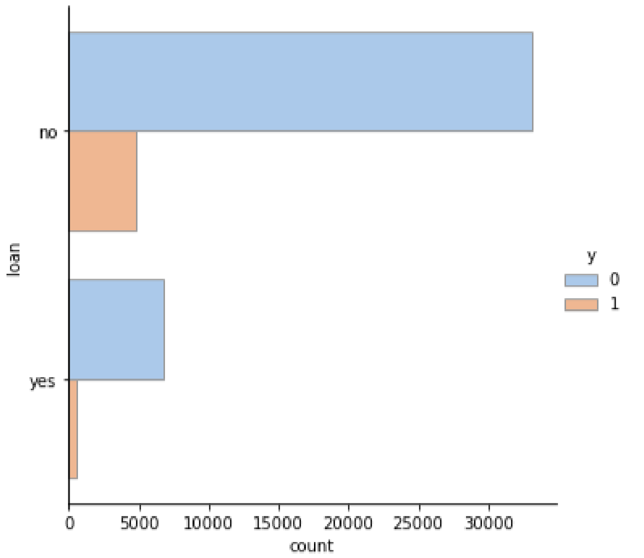


People with no housing scheme have subscribed for the term deposit.

➢ Term deposit based on Personal Loan

```
sns.catplot(
    data=d, y="loan", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d2f1a3cf70>
```

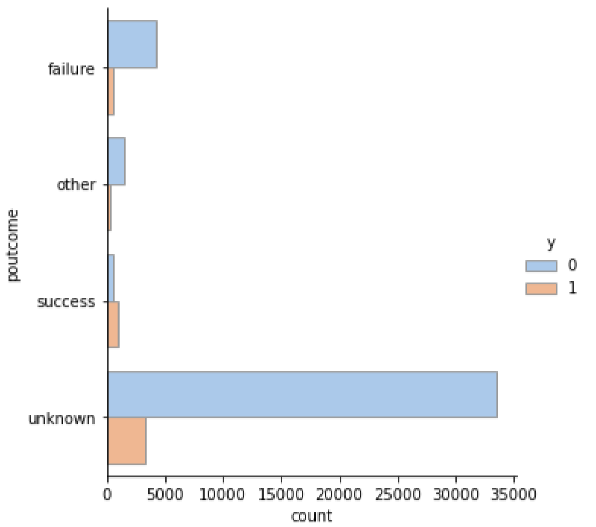


People with no personal loan are more interested in term deposit.

➢ Term deposit based on outcome of Previous Campaign

```
sns.catplot(
    data=d, y="poutcome", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d9864430a0>
```



From the Outcome of previous Campaign, if the outcome is Failure, then there is a less chance that client will subscribe to the term deposit. whereas if the outcome of previous Campaign is Success, then it is more likely that Client will subscribe to the term deposit.

10. Transformation

10.1. Data set information

```
# Datatypes of columns and non-null values
d.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

As per dataset information, the data types of columns are integer and object, but we know object columns are categorical columns so, first we convert the data types of categorical columns into 'category.'

10.2. Convert datatype.

```
# change datatype of categorical columns into "category"
d["job"]=d["job"].astype("category")
d["marital"]=d["marital"].astype("category")
d["education"]=d["education"].astype("category")
d["default"]=d["default"].astype("category")
d["housing"]=d["housing"].astype("category")
d["loan"]=d["loan"].astype("category")
d["contact"]=d["contact"].astype("category")
d["month"]=d["month"].astype("category")
d["poutcome"]=d["poutcome"].astype("category")
d["y"]=d["y"].astype("category")
```

```
d.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype   
---  ------     --------------  -----   
 0   age        45211 non-null  int64   
 1   job        45211 non-null  category
 2   marital    45211 non-null  category
 3   education  45211 non-null  category
 4   default    45211 non-null  category
 5   balance    45211 non-null  int64   
 6   housing    45211 non-null  category
 7   loan       45211 non-null  category
 8   contact    45211 non-null  category
 9   day        45211 non-null  int64   
 10  month      45211 non-null  category
 11  campaign   45211 non-null  int64   
 12  pdays      45211 non-null  int64   
 13  previous   45211 non-null  int64   
 14  poutcome   45211 non-null  category
 15  y          45211 non-null  category
dtypes: category(10), int64(6)
memory usage: 2.8 MB
```

Here we see the data types of all categorical column is 'category.'

10.3. Encoding – Label encoding

All machine learning algorithms work with only numerical values so, second transformation that is needed to be done is to convert all categorical columns into numerical columns. Here we use label encoding technique for conversion.

```
from sklearn import preprocessing
```

```
le=preprocessing.LabelEncoder()
d['job']=le.fit_transform(d['job'])
d['marital']=le.fit_transform(d['marital'])
d['education']=le.fit_transform(d['education'])
d['default']=le.fit_transform(d['default'])
d['housing']=le.fit_transform(d['housing'])
d['loan']=le.fit_transform(d['loan'])
d['contact']=le.fit_transform(d['contact'])
d['month']=le.fit_transform(d['month'])
d['poutcome']=le.fit_transform(d['poutcome'])
```

```
d.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	y
0	58	4	1	2	0	2143	1	0	2	5	8	1	-1	0	3	0
1	44	9	2	1	0	29	1	0	2	5	8	1	-1	0	3	0
2	33	2	1	1	0	2	1	1	2	5	8	1	-1	0	3	0
3	47	1	1	3	0	1506	1	0	2	5	8	1	-1	0	3	0
4	33	11	2	3	0	1	0	0	2	5	8	1	-1	0	3	0

11. Feature Scaling -Normalization

To deal with noises in the data, we need to perform feature scaling and as there are both continuous and discrete columns, we are using **normalization scaling technique** to transform features to be on a similar scale. **This improves the performance and training stability of the model**.

```
In [67]: from numpy import set_printoptions
         from sklearn.preprocessing import MinMaxScaler

In [68]: d1=d.iloc[:,:-1]
         d1
```

Out[68]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome
0	58	4	1	2	0	2143	1	0	2	5	8	1	-1	0	3
1	44	9	2	1	0	29	1	0	2	5	8	1	-1	0	3
2	33	2	1	1	0	2	1	1	2	5	8	1	-1	0	3
3	47	1	1	3	0	1506	1	0	2	5	8	1	-1	0	3
4	33	11	2	3	0	1	0	0	2	5	8	1	-1	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	51	9	1	2	0	825	0	0	0	17	9	3	-1	0	3
45207	71	5	0	0	0	1729	0	0	0	17	9	2	-1	0	3
45208	72	5	1	1	0	5715	0	0	0	17	9	5	184	3	2
45209	57	1	1	1	0	668	0	0	1	17	9	4	-1	0	3
45210	37	2	1	1	0	2971	0	0	0	17	9	2	188	11	1

45211 rows × 15 columns

```
In [69]: array=d1.values
         scaler=MinMaxScaler(feature_range=(0,1))
         rescaledX=scaler.fit_transform(array)

         set_printoptions(precision=2)
         print(rescaledX[0:5,:])

[[0.52 0.36 0.5  0.67 0.   0.09 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.34 0.82 1.   0.33 0.   0.07 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.19 0.18 0.5  0.33 0.   0.07 1.   1.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.38 0.09 0.5  1.   0.   0.09 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.19 1.   1.   1.   0.   0.07 0.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]]
```

```
In [70]: d2=pd.DataFrame(rescaledX,columns=["age","job","marital","education","default","balance","housing","loan","contact","day",
                                            "month","campaign","pdays","previous","poutcome"])
         d2
```

Out[70]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome
0	0.519481	0.363636	0.5	0.666667	0.0	0.092259	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
1	0.337662	0.818182	1.0	0.333333	0.0	0.073067	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
2	0.194805	0.181818	0.5	0.333333	0.0	0.072822	1.0	1.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
3	0.376623	0.090909	0.5	1.000000	0.0	0.086476	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
4	0.194805	1.000000	1.0	1.000000	0.0	0.072812	0.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	0.428571	0.818182	0.5	0.666667	0.0	0.080293	0.0	0.0	0.0	0.533333	0.818182	0.032258	0.000000	0.000000	1.000000
45207	0.688312	0.454545	0.0	0.000000	0.0	0.088501	0.0	0.0	0.0	0.533333	0.818182	0.016129	0.000000	0.000000	1.000000
45208	0.701299	0.454545	0.5	0.333333	0.0	0.124689	0.0	0.0	0.0	0.533333	0.818182	0.064516	0.212156	0.010909	0.666667
45209	0.506494	0.090909	0.5	0.333333	0.0	0.078868	0.0	0.0	0.5	0.533333	0.818182	0.048387	0.000000	0.000000	1.000000
45210	0.246753	0.181818	0.5	0.333333	0.0	0.099777	0.0	0.0	0.0	0.533333	0.818182	0.016129	0.216743	0.040000	0.333333

45211 rows × 15 columns

12. Model Building and Model Selection

12.1. Balance the dataset

```
X=d2
Y=d.iloc[:,-1]
```

```
Y.value_counts()

0    39922
1     5289
Name: y, dtype: int64
```

The dataset is imbalance, so we will balance the dataset using SMOTE.

```
# balance the dataset using SMOTE
from imblearn.over_sampling import SMOTE
from collections import Counter

sm = SMOTE(random_state=42)
X_res, Y_res = sm.fit_resample(X, Y)
print('Resampled dataset shape %s' % Counter(Y_res))

Resampled dataset shape Counter({0: 39922, 1: 39922})

Y_res.value_counts()

0    39922
1    39922
Name: y, dtype: int64
```

12.2. Split dataset into Train and Test datasets

Import train_test_split and divide the dataset into input variables and output variable then split the input and output into train and test sets (30% test and 70% train).

```
from sklearn.model_selection import train_test_split
```

```
# splitting dataset in 70% train dataset and 30% test dataset
X_train,X_test,Y_train,Y_test =train_test_split(X_res,Y_res, test_size=0.3,random_state=0)

X_train.shape

(55890, 15)

X_test.shape

(23954, 15)
```

12.3. Logistic Regression Model

After data pre-processing, a machine learning model is created to predict term deposit subscription. For this purpose, Logistic regression algorithm is used from sklearn. linear_model. After importing and initialize Logistic Regression model the dataset is being fitted for training using classifier.

```
from sklearn.linear_model import LogisticRegression

classifier=LogisticRegression()
classifier.fit(X_train,Y_train) # Fit the model to the training data

LogisticRegression()

Y_pred=classifier.predict(X_test) # Predict the classes on the test data
Y_pred

array([1, 1, 0, ..., 1, 1, 1])

np.mean(Y_pred==Y_test)

0.6629790431660683

pd.crosstab(Y_test,Y_pred)
```

col_0	0	1
y		
0	7288	4501
1	3572	8593

```
lreg_data=classifier.score(X,Y)
lreg_train=classifier.score(X_train,Y_train)
lreg_test=classifier.score(X_test,Y_test)

print ("Accuracy of All dataset: " ,(lreg_data))
print ("Accuracy of Train dataset: " ,(lreg_train))
print ("Accuracy of Test dataset: " ,(lreg_test))

Accuracy of All dataset:  0.6300236668067506
Accuracy of Train dataset:  0.6628556092324208
Accuracy of Test dataset:  0.6629790431660683
```

The score of the Logistic Regression model is quite low and its under-fitting model. Let's train the model with another algorithm.

12.4. Random Forest Classifier

Random Forest classification algorithm is used from sklearn. ensemble. After importing and initialize Random Forest classification model the dataset is being fitted for training using clf.

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier(max_depth=3, random_state=42)
clf.fit(X_train,Y_train) # Fit the model to the training data
```

```
RandomForestClassifier(max_depth=3, random_state=42)
```

```
Y1_pred=clf.predict(X_test) # Predict the classes on the test data
Y1_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
np.mean(Y1_pred==Y_test)
```

```
0.7181681556316273
```

```
pd.crosstab(Y_test,Y1_pred)
```

col_0	0	1
y		
0	9245	2544
1	4207	7958

```
rft_data=clf.score(X,Y)
rft_train=clf.score(X_train,Y_train)
rft_test=clf.score(X_test,Y_test)
```

```
print ("Accuracy of All dataset: " ,(rft_data))
print ("Accuracy of Train dataset: " ,(rft_train))
print ("Accuracy of Test dataset: " ,(rft_test))
```

```
Accuracy of All dataset:  0.760766185220411
Accuracy of Train dataset:  0.7219180533190195
Accuracy of Test dataset:  0.7181681556316273
```

The Random Forest classification model score is better than the Logistic Regression model but there are so many false positives and false negatives now we try another algorithm to train the model that is Gradient Boosting Classifier.

12.5. Gradient Boosting Classifier

Gradient Boosting classification algorithm is used from sklearn. ensemble. After importing and initialize Gradient Boosting classification model the dataset is being fitted for training using model.

```
from sklearn.ensemble import GradientBoostingClassifier

model=GradientBoostingClassifier(n_estimators=300, learning_rate=1.0, max_depth=2, random_state=40)
model.fit(X_train,Y_train) # Fit the model to the training data

GradientBoostingClassifier(learning_rate=1.0, max_depth=2, n_estimators=300,
                           random_state=40)

Y2_pred=model.predict(X_test) # Predict the classes on the test data
Y2_pred

array([1, 1, 1, ..., 0, 1, 1])

np.mean(Y2_pred==Y_test)

0.9336645236703682

pd.crosstab(Y_test,Y2_pred)
```

col_0	0	1
y		
0	11442	347
1	1242	10923

```
gbc_data=model.score(X,Y)
gbc_train=model.score(X_train,Y_train)
gbc_test=model.score(X_test,Y_test)

print ("Accuracy of All dataset: " ,(gbc_data))
print ("Accuracy of Train dataset: " ,(gbc_train))
print ("Accuracy of Test dataset: " ,(gbc_test))

Accuracy of All dataset:  0.8989405233239698
Accuracy of Train dataset:  0.9436750760422258
Accuracy of Test dataset:  0.9336645236703682
```

Gradient Boosting Classifier has the best score so far. Next step is to perform hyperparameter tuning to try to improve accuracy of the model.

12.6. Hyper-parameter Tuning

Accuracy of Gradient Boosting Classifier is better than other models but here we have more false positives and false negative so we will do hyperparameter tunning of Gradient Boosting Classifier model. For hyperparameter tunning, Grid CV search from sklearn. model_selection will be used.

```
from sklearn.model_selection import GridSearchCV

gb = GradientBoostingClassifier()
parameters = {
    "n_estimators":[5,50,75],
    "max_depth":[1,3,5],
    "learning_rate":[0.01,0.1,1]
}

cv = GridSearchCV(gb,parameters,cv=5) # Here we are using 5 iterations
cv.fit(X_train,Y_train)

GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'learning_rate': [0.01, 0.1, 1],
                         'max_depth': [1, 3, 5], 'n_estimators': [5, 50, 75]})

cv.best_params_

{'learning_rate': 1, 'max_depth': 3, 'n_estimators': 75}

cv.best_score_

0.9309536589729828
```

Now train the model with best parameters obtained after hyperparameter tuning. Gradient Boosting classification model the dataset is being fitted for training using model1.

```
model1=GradientBoostingClassifier(n_estimators=75, learning_rate=1, max_depth=3, random_state=42)
model1.fit(X_train,Y_train)
```

```
GradientBoostingClassifier(learning_rate=1, n_estimators=75, random_state=42)
```

```
YY_pred=model1.predict(X_test)
YY_pred
```

```
array([1, 1, 1, ..., 0, 1, 1])
```

```
np.mean(YY_pred==Y_test)
```

```
0.9297403356433164
```

```
pd.crosstab(Y_test,YY_pred)
```

col_0	0	1
y		
0	11385	404
1	1279	10886

```
gb_data=model1.score(X,Y)
gb_train=model1.score(X_train,Y_train)
gb_test=model1.score(X_test,Y_test)
```

```
print ("Accuracy of All dataset: " ,(gb_data))
print ("Accuracy of Train dataset: " ,(gb_train))
print ("Accuracy of Test dataset: " ,(gb_test))
```

```
Accuracy of All dataset:  0.8957996947645485
Accuracy of Train dataset:  0.9386652352835928
Accuracy of Test dataset:  0.9297403356433164
```

After Hyperparameter tuning the accuracy has reduced so we choose Gradient Boosting classifier model without Hyperparameter Tuning

12.7. Metrics for Evaluation

12.7.1. Accuracy, Precision, Recall and F1-Score

```
from sklearn.metrics import classification_report
```

- Logistic Regression model

```
#LogisticRegression
resultsL=classifier.score(X,Y)
resultsL
```

```
0.6300236668067506
```

- Random Forest Classifier

```
#RandomForestTresClassifier
print(classification_report(Y_test,Y1_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.78      0.73     11789
           1       0.76      0.65      0.70     12165

    accuracy                           0.72     23954
   macro avg       0.72      0.72      0.72     23954
weighted avg       0.72      0.72      0.72     23954
```

- Gradient Boosting Classifier with hyperparameter tuning.

```
#GradientBoostingClassifier with parameter tuning
print(classification_report(Y_test,YY_pred))

              precision    recall  f1-score   support

           0       0.90      0.97      0.93     11789
           1       0.96      0.89      0.93     12165

    accuracy                           0.93     23954
   macro avg       0.93      0.93      0.93     23954
weighted avg       0.93      0.93      0.93     23954
```

- Gradient Boosting Classifier without hyperparameter tuning.

```
#GradientBoostingClassifier without parameter tuning
print(classification_report(Y_test,Y2_pred))

              precision    recall  f1-score   support

           0       0.90      0.97      0.94     11789
           1       0.97      0.90      0.93     12165

    accuracy                           0.93     23954
   macro avg       0.94      0.93      0.93     23954
weighted avg       0.94      0.93      0.93     23954
```

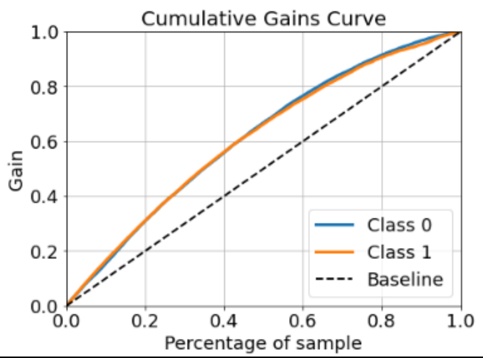
Based on Classification report Gradient Boosting Classifier model without Hyperparameter tuning is the best model.

12.7.2. Lift and Gain

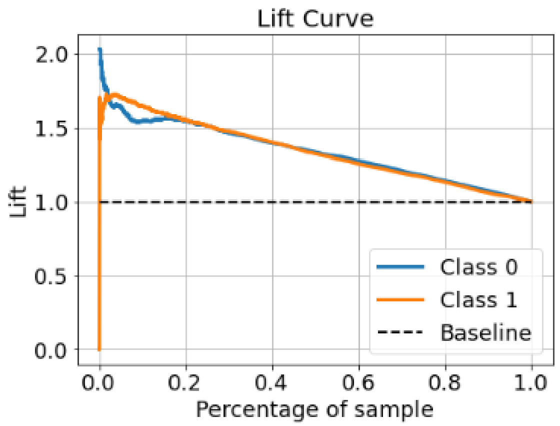
```
import scikitplot as skplt
```

- Logistic Regression Model

```
# Predict the classes on the test data, and return the probabilities for each class
Y_proba = classifier.predict_proba(X_test)
skplt.metrics.plot_cumulative_gain(Y_test, Y_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```

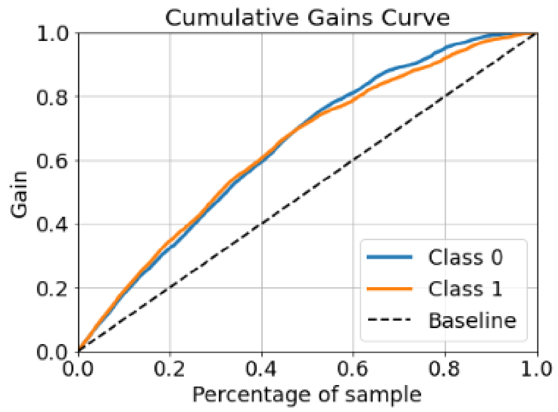


```
skplt.metrics.plot_lift_curve(Y_test, Y_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```

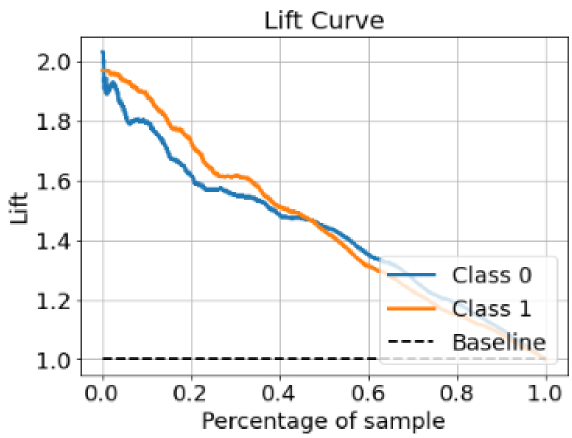


- Random Forest Classification Model

```
# Predict the classes on the test data, and return the probabilities for each class
Y1_proba = clf.predict_proba(X_test)
skplt.metrics.plot_cumulative_gain(Y_test, Y1_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```

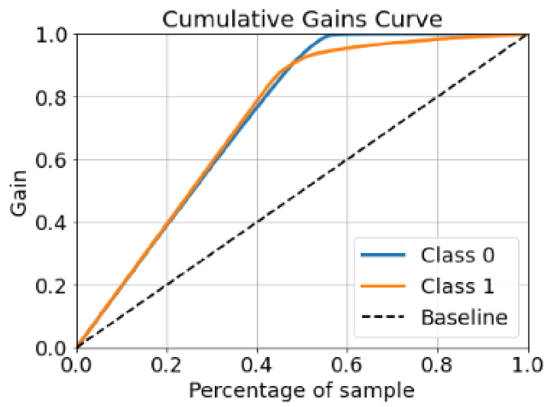



```
skplt.metrics.plot_lift_curve(Y_test, Y1_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```

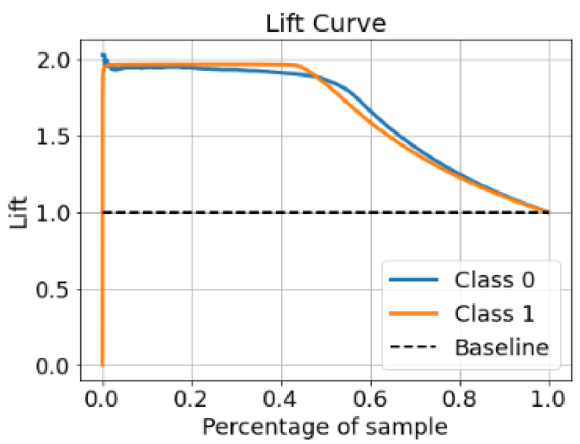


➢ Gradient Boosting Classification model

```
# Predict the classes on the test data, and return the probabilities for each class
Y2_proba = model.predict_proba(X_test)
skplt.metrics.plot_cumulative_gain(Y_test, Y2_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```




```
skplt.metrics.plot_lift_curve(Y_test, Y2_proba, figsize=(7, 5), title_fontsize=20, text_fontsize=18)
plt.show()
```



Cumulative gains and lift charts are visual aids for measuring model performance.
The Greater the area between the Lift / Gain and Baseline, the Better the model.
By analysing Gain and Lift Curve, Gradient Boosting Classifier is the best model.

12.7.3. KS Statistics and ROC-AUC Score

In most binary classification problems, we use the KS-2samp test and ROC AUC score as measurements of how well the model separates the predictions of the two different classes.
The KS statistic for two samples is simply the highest distance between their two CDFs, so if we measure the distance between the positive and negative class distributions, we can have another metric to evaluate classifiers.
The ROC AUC score goes from 0.5 to 1.0, while KS statistics range from 0.0 to 1.0.

```
from scipy import stats
from scipy.stats import ks_2samp
from sklearn.metrics import roc_auc_score
```

```python
def evaluate_ks_and_roc_auc(y_real, y_proba):
    # Unite both visions to be able to filter
    df = pd.DataFrame()
    df['real'] = y_real
    df['proba'] = y_proba[:, 1]

    # Recover each class
    class0 = df[df['real'] == 0]
    class1 = df[df['real'] == 1]

    ks = ks_2samp(class0['proba'], class1['proba'])
    roc_auc = roc_auc_score(df['real'] , df['proba'])

    print(f"KS: {ks.statistic:.4f} (p-value: {ks.pvalue:.3e})")
    print(f"ROC AUC: {roc_auc:.4f}")

    return ks.statistic, roc_auc
```

- Logistic Regression Model

```python
#Logistic Regression
# Fit the model to the training data
classifier.fit(X_train,Y_train)
# Predict the classes on the test data
Y_pred=classifier.predict(X_test)
# Predict the classes on the test data, and return the probabilities for each class
Y_proba = classifier.predict_proba(X_test)
```

```python
print("Logistic Regression:")
ks_LR, auc_LR = evaluate_ks_and_roc_auc(Y_test, Y_proba)
```

```
Logistic Regression:
KS: 0.3360 (p-value: 0.000e+00)
ROC AUC: 0.7246
```

- Random Forest Classification Model

```python
#RandomForestClassifier
# Fit the model to the training data
clf.fit(X_train,Y_train)
# Predict the classes on the test data
Y1_pred=clf.predict(X_test)
# Predict the classes on the test data, and return the probabilities for each class
Y1_proba = clf.predict_proba(X_test)
```

```python
print("Random Forest classifier:")
ks_RFC, auc_RFC = evaluate_ks_and_roc_auc(Y_test, Y1_proba)
```

```
Random Forest classifier:
KS: 0.4475 (p-value: 0.000e+00)
ROC AUC: 0.7850
```

➢ Gradient Boosting Classification Model

```
#BoostingGradientClassifier
# Fit the model to the training data
model.fit(X_train,Y_train)
# Predict the classes on the test data
Y2_pred=model.predict(X_test)
# Predict the classes on the test data, and return the probabilities for each class
Y2_proba = model.predict_proba(X_test)
```

```
print("Gradient Boosting classifier:")
ks_GBC, auc_GBC = evaluate_ks_and_roc_auc(Y_test, Y2_proba)

Gradient Boosting classifier:
KS: 0.8697 (p-value: 0.000e+00)
ROC AUC: 0.9690
```

Gradient Boosting Classifier has got ROC AUC of 0.9690 which is almost perfect and KS score is 0.8697 which reflects better the fact that the classes are not "almost perfectly" separable.

13. Save the Model

Last step is saving the model using pickle.

```
# import pickle library
import pickle # its used for seriealizing and de-seriealizing a python object Structure
pickle.dump(model, open('model.pkl','wb'))       # open the file for writing
model = pickle.load(open('model.pkl','rb'))     # dump an object to file object
```

14. Deployment of model into flask framework

A web application is developed that consists of a web page, after submitting the input in the form-based field to the web application, it will give the predicted Term deposit subscription. Following is the directory structure of all files used for application.

14.1. App.py

The app.py file contains the source code including the ML code for prediction and will be execute by the Python interpreter to run the Flask web application.

```python
import numpy as np
from flask import Flask, request, render_template
import pickle

#Create an app object using the Flask class.
app = Flask(__name__)

#Load the trained model. (Pickle file)
model = pickle.load(open('models/model.pkl', 'rb'))

#Define the route to be home.
#The decorator below links the relative route of the URL to the function it is decorating.
#Here, home function is with '/', our root directory.
#Running the app sends us to index.html.
#Note that render_template means it looks for the file in the templates folder.

#use the route() decorator to tell Flask what URL should trigger our function.
@app.route('/')
def home():
  return render_template('index.html')

#You can use the methods argument of the route() decorator to handle different HTTP methods.
#GET: A GET message is send, and the server returns data
#POST: Used to send HTML form data to the server.
#Add Post method to the decorator to allow for form submission.
#Redirect to /predict page with the output
@app.route('/predict',methods=['POST'])
def predict():

    int_features = [float(x) for x in request.form.values()] #Convert string inputs to float.
    features = [np.array(int_features)]  #Convert to the form [[a, b,c]] for input to the model
    prediction = model.predict(features)  # features Must be in the form [[a, b,c]]

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text="{}".format(output))


#When the Python interpreter reads a source file, it first defines a few special variables.
#For now, we care about the __name__ variable.
#If we execute our code in the main program, like in our case here, it assigns
# __main__ as the name (__name__).
#So if we want to run our code right here, we can check if __name__ == __main__
#if so, execute it here.
#If we import this file (module) to another file then __name__ == app (which is the name of this python file).

if __name__ == "__main__":
    app.run()
```

▪ Application will run as a single module; thus, a new Flask instance is initialized with the argument __name__ to let Flask know that it can find the HTML template folder (templates) in the same directory where it is located.

▪ Next, the route decorator is used (@app. route ('/')) to specify the URL that should trigger the execution of the home function. Home function simply rendered the index.html HTML file, which is in the templates folder.

▪ Predict function has the data set, it pre-processes the input, and make predictions, and then store the model. The input is entered by the user and uses the model to make a prediction for its label.

▪ The POST method is used to transport the form data to the server in the message body.

▪ The run function is used to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the if statement with __name__ == '__main__'.

14.2. Index.html

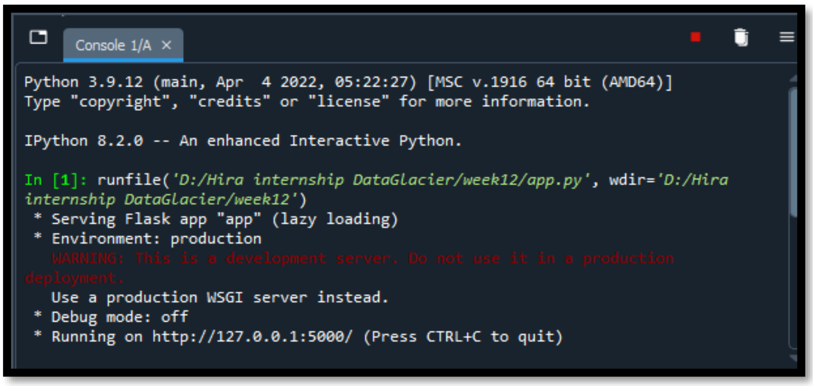
The Index.html file will render a text form where a user enter the details of required fields. Index.html file will be rendered via the render_template ('index.html', prediction_text="{}".format(output)), which is inside the predict function of app.py script to display the output as per the input submitted by the user.

```html
<!DOCTYPE html>

<html >
<head>
  <meta charset="UTF-8">

  <!-- Make it compatible to mobile devices -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>BANK MARKETING CAMPAIGN</title>

</head>

<body style="background: #99C68E;">
 <div class="login">
    <h1 style="color:Black;">PREDICTION FOR CUSTOMER INTEREST IN  DEPOSITE TERM</h1>

     <!-- Action is where the data is sent. In our case, predict page.
     If action is omitted, it assumed to be the current page -->
    <form action="{{ url_for('predict')}}"method="post">
    <h8 style="color:Red;">Note: All variables are normalised.Please enter normalised values(Range: 0-1)</h8><br></br>
        <label for="Age">Age:</label>
        <input type="text" name="Age" placeholder="Enter your Age" required="required" /><p><br>
        <label for="Job">Job:</label>
        <input type="text" name="Job" placeholder="Select option " required="required" /><p>
        <h8 style="color:white;">List of Job:</h8><br>
        <h8 style="color:white;">0.00:Admin, 0.091:Blue-Collar, 0.182:Entrepreneur, 0.273:Housemaid,
        0.364:Mangement, 0.454:Retired,</h8><br>
        <h8 style="color:white;"> 0.545:Self-employed,0.636:Services, 0.727:student, 0.818:Technician,
        0.91:Unemployed, 1.00:Un-known</h8><br></br>
        <label for="Marital">Marital Status:</label>
        <input type="text" name="Marital" placeholder="Select option" required="required" /><p>
        <h8 style="color:white;">Marital Status:</h8>
        <h8 style="color:white;">0.0:Divorced/widow, 0.5:Married, 1.0:Single</h8><br></br>
        <label for="Education">Education:</label>
        <input type="text" name="Education" placeholder="Select option" required="required" /><p>
        <h8 style="color:white;">Level of Education:</h8>
        <h8 style="color:white;">0.0:Primary, 0.33:Secondary, 0.67:Tertiary, 1.0:Unknown</h8><br></br>
        <label for="Default">Credit Default:</label>
        <input type="text" name="Default" placeholder="0.0=No / 1.0=Yes" required="required" /><p><br>
        <label for="Balance">Balance:</label>
```

```html
        <input type="text" name="Balance" placeholder="Enter Balance" required="required" /><p><br>
        <label for="Housing">Housing Loan:</label>
        <input type="text" name="Housing" placeholder="0.0=No / 1.0=Yes" required="required" /><p><br>
        <label for="Loan">Personal Loan:</label>
        <input type="text" name="Loan" placeholder="0.0=No / 1.0=Yes" required="required" /><p><br>
        <label for="Contact">Contact Communicattion Type:</label>
        <input type="text" name="Contact" placeholder="Select option" required="required" /><p>
        <h8 style="color:white;">Contact Communicattion Types:</h8>
        <h8 style="color:white;">0.0:Cellular, 0.5:Telephone, 1.0:Unknown</h8><br></br>
        <label for="Day">Day:</label>
        <input type="text" name="Day" placeholder="Enter workday number" required="required" /><p><br>
        <label for="Month">Month:</label>
        <input type="text" name="Month" placeholder="Enter month number" required="required" /><p>
        <h8 style="color:white;">Months:</h8><br>
        <h8 style="color:white;">0.364:Jan, 0.273:Feb, 0.636:Mar, 0.00:April, 0.727:May, 0.545:jun,
         0.454:Jul, 0.091:Aug, 1.00:Sep, 0.91:Oct, 0.818:Nov, 0.182:Dec</h8><br></br>
        <label for="Campaign">Campaign:</label>
        <input type="text" name="Campaign" placeholder="Enter campaign days" required="required" /><p><br>
        <label for="Pdays">Days Passed by:</label>
        <input type="text" name="Pdays" placeholder="Enter number of days" required="required" /><p><br>
        <label for="Previous">Previous:</label>
        <input type="text" name="Previous" placeholder="Enter number of days"" required="required" /><p><br>
        <label for="Poutcome">Previous Marketing Outcome:</label>
        <input type="text" name="Poutocme" placeholder="Select option" required="required" /><p>
        <h8 style="color:white;">Previous Marketing Outcome:</h8>
        <h8 style="color:white;">0.00:Failure, 0.667:Success, 0.33:Other, 1.00:Unknown</h8><br></br>

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button></p>
    </form>

 </div>
    <div class="login">

    <h1 style="color:Red;">The client will subscribe a term deposit: {{prediction_text}}</h1>
    </div>
    <h8 style="color:white;">1 = Yes </h8><br>
    <h8 style="color:white;">0 = No </h8><br>
</body>
</html>
```

14.3. Development Server

Following is the URL generate by 'app.py.'



Now open a web browser and navigate to http://127.0.0.1:5000/ following is output of Index.html.

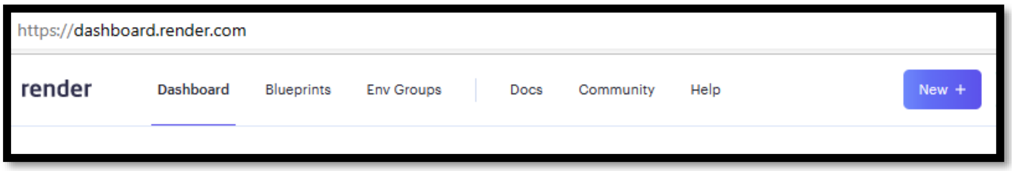
Fill in the required fields with normalised input values. Select categorical fields as per their respective number in the given code and click the Predict button. The predicted result will be displayed at the bottom of the web page.


127.0.0.1:5000
PREDICTION FOR CUSTOMER INTEREST IN DEPOSITE TERM
Note: All variables are nomalised.Please enter normalised values(Range: 0-1)
Age: 0.5
Job: 1.0
List of Job:
0.00:Admin, 0.091:Blue-Collar, 0.182:Entrepreneur, 0.273:Housemaid, 0.364:Mangement, 0.454:Retired,
0.545:Self-employed,0.636:Services, 0.727:student, 0.818:Technician, 0.91:Unemployed, 1.00:Un-known
Marital Status: 0.5
Marital Status: 0.0:Divorced/widow, 0.5:Married, 1.0:Single
Education: 0.667
Level of Education: 0.0:Primary, 0.33:Secondary, 0.67:Tertiary, 1.0:Unknown
Credit Default: 0.0
Balance: 0.09
Housing Loan: 1.0
Personal Loan: 0.0
Contact Communicattion Type: 1.0
Contact Communicattion Types: 0.0:Cellular, 0.5:Telephone, 1.0:Unknown
Day: 0.133
Month: 0.727
Months:
0.364:Jan, 0.273:Feb, 0.636:Mar, 0.00:April, 0.727:May, 0.545:jun, 0.454:Jul, 0.091:Aug, 1.00:Sep, 0.91:Oct, 0.818:Nov, 0.182:Dec
Campaign: 0.0
Days Passed by: 0.0
Previous: 0.0
Previous Marketing Outcome: 1.0
Previous Marketing Outcome: 0.00:Failure, 0.667:Success, 0.33:Other, 1.00:Unknown
Predict
The client will subscribe a term deposit:
1 = Yes
0 = No


127.0.0.1:5000/predict
PREDICTION FOR CUSTOMER INTEREST IN DEPOSITE TERM
Note: All variables are nomalised.Please enter normalised values(Range: 0-1)
Age: Enter your Age
Job: Select option
List of Job:
0.00:Admin, 0.091:Blue-Collar, 0.182:Entrepreneur, 0.273:Housemaid, 0.364:Mangement, 0.454:Retired,
0.545:Self-employed,0.636:Services, 0.727:student, 0.818:Technician, 0.91:Unemployed, 1.00:Un-known
Marital Status: Select option
Marital Status: 0.0:Divorced/widow, 0.5:Married, 1.0:Single
Education: Select option
Level of Education: 0.0:Primary, 0.33:Secondary, 0.67:Tertiary, 1.0:Unknown
Credit Default: 0.0=No / 1.0=Yes
Balance: Enter Balance
Housing Loan: 0.0=No / 1.0=Yes
Personal Loan: 0.0=No / 1.0=Yes
Contact Communicattion Type: Select option
Contact Communicattion Types: 0.0:Cellular, 0.5:Telephone, 1.0:Unknown
Day: Enter workday number
Month: Enter month number
Months:
0.364:Jan, 0.273:Feb, 0.636:Mar, 0.00:April, 0.727:May, 0.545:jun, 0.454:Jul, 0.091:Aug, 1.00:Sep, 0.91:Oct, 0.818:Nov, 0.182:Dec
Campaign: Enter campaign days
Days Passed by: Enter number of days
Previous: Enter number of days
Previous Marketing Outcome: Select option
Previous Marketing Outcome: 0.00:Failure, 0.667:Success, 0.33:Other, 1.00:Unknown
Predict
The client will subscribe a term deposit: 1
1 = Yes
0 = No

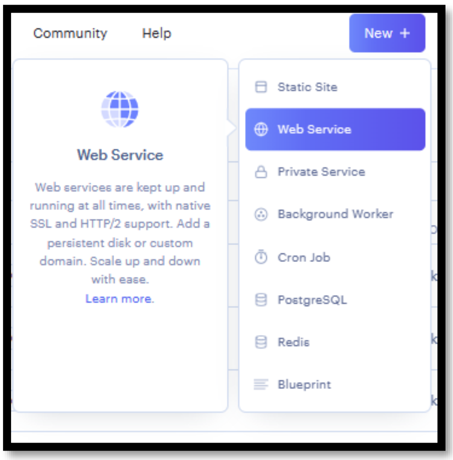
15. Model deployment on Render (Open-Source Cloud Deployment)

After the model has been trained and deployed locally, now it is ready for deploy on open-source cloud "Render".



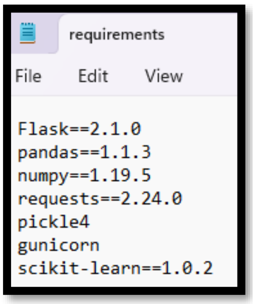
15.1. Web Service

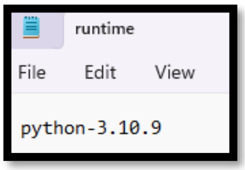
Click 'New +' then select 'Web Service.'

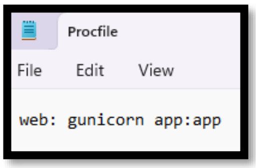


15.2. Connect to GitHub Repository

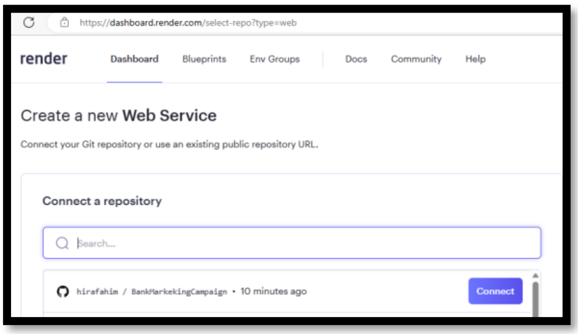
Before connecting to the GitHub repository, add required packages to the GitHub repository.

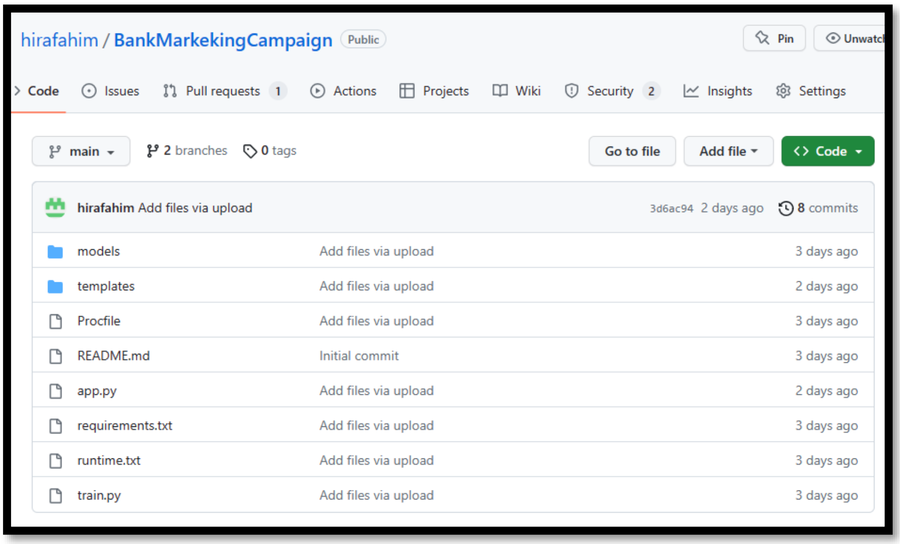




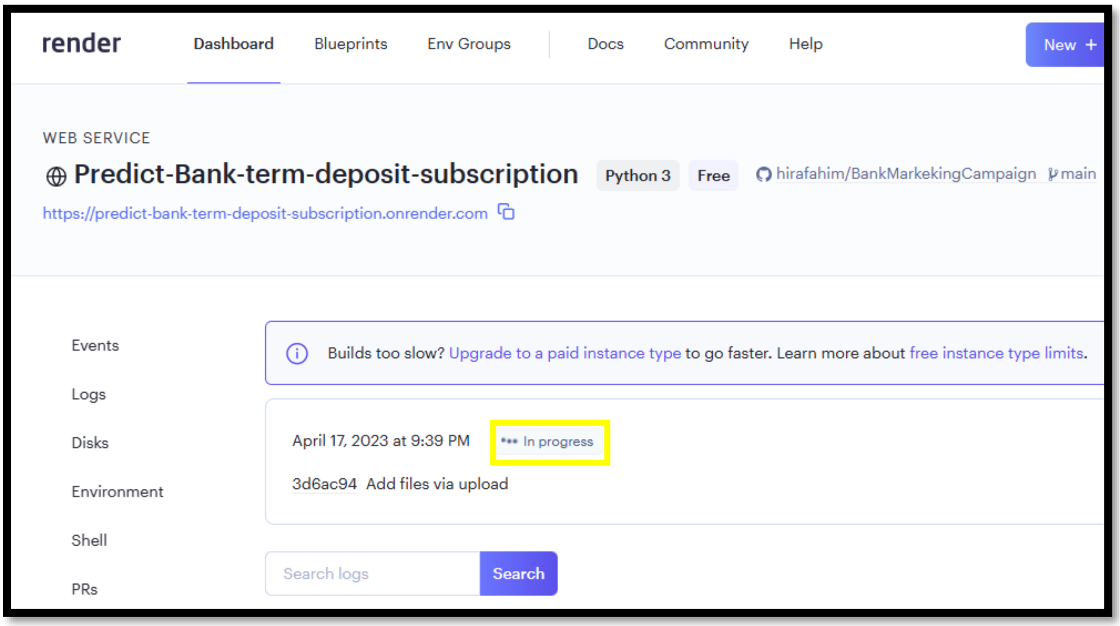


Connect to the GitHub repository.

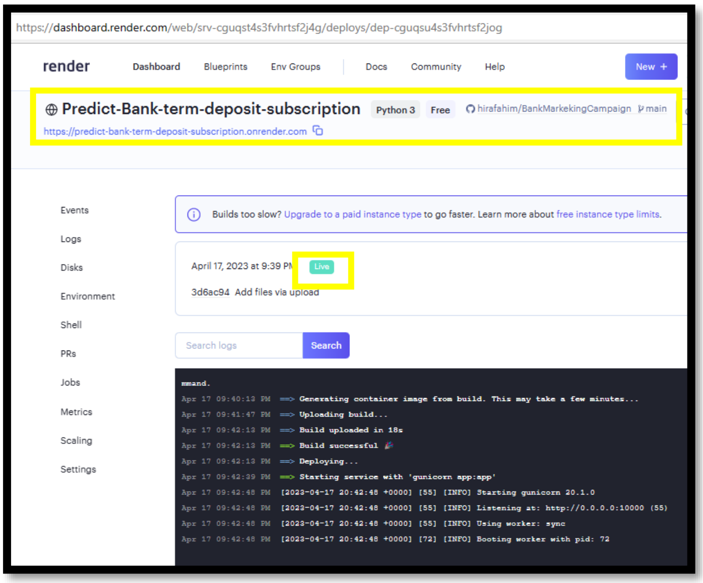


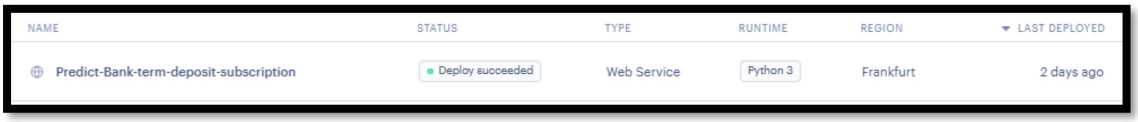


Fill in the required fields on Render dashboard and click the create to deploy the web app.



After 10 minutes, the web app is built and deployed successfully.



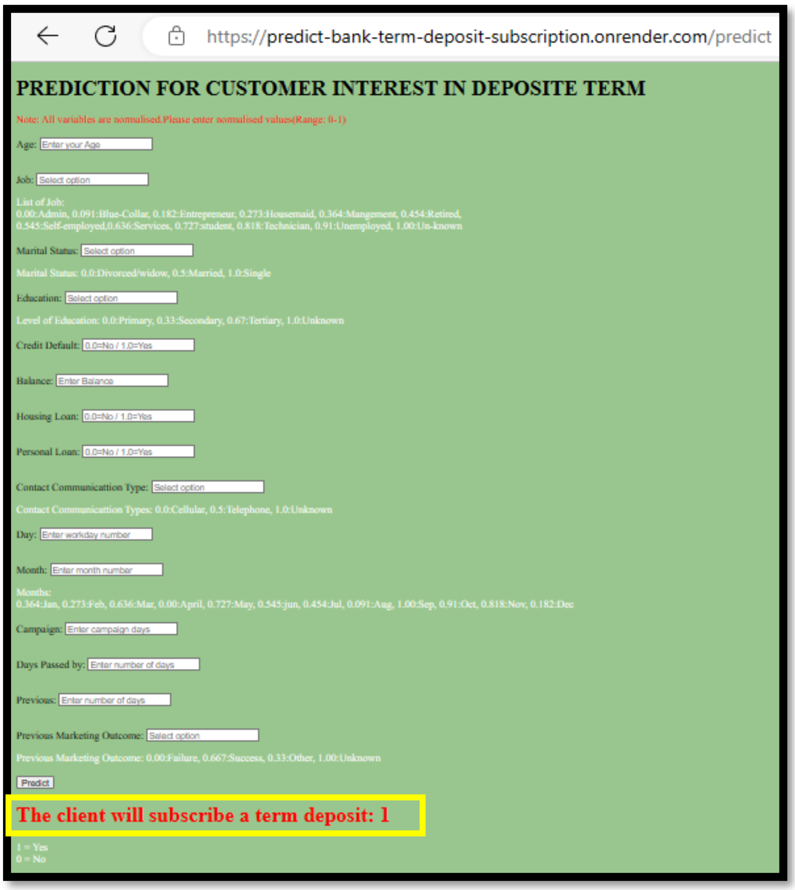


15.3. API- User Interface

This is the website link, click and open the application for prediction of Term deposit subscription.

https://predict-bank-term-deposit-subscription.onrender.com


https://predict-bank-term-deposit-subscription.onrender.com
PREDICTION FOR CUSTOMER INTEREST IN DEPOSITE TERM
Note: All variables are normalised.Please enter normalised values(Range: 0-1)
Age: 1.0
Job: 0.364
List of Job:
0.00:Admin, 0.091:Blue-Collar, 0.182:Entrepreneur, 0.273:Housemaid, 0.364:Mangement, 0.454:Retired,
0.545:Self-employed,0.636:Services, 0.727:student, 0.818:Technician, 0.91:Unemployed, 1.00:Un-known
Marital Status: 0.0
Marital Status: 0.0:Divorced/widow, 0.5:Married, 1.0:Single
Education: 0.33
Level of Education: 0.0:Primary, 0.33:Secondary, 0.67:Tertiary, 1.0:Unknown
Credit Default: 0.0
Balance: 0.09
Housing Loan: 0.0
Personal Loan: 0.0
Contact Communicattion Type: 0.5
Contact Communicattion Types: 0.0:Cellular, 0.5:Telephone, 1.0:Unknown
Day: 0.67
Month: 0.454
Months:
0.364:Jan, 0.273:Feb, 0.636:Mar, 0.00:April, 0.727:May, 0.545:jun, 0.454:Jul, 0.091:Aug, 1.00:Sep, 0.91:Oct, 0.818:Nov, 0.182:Dec
Campaign: 0.0
Days Passed by: 0.0
Previous: 0.0
Previous Marketing Outcome: 1.0
Previous Marketing Outcome: 0.00:Failure, 0.667:Success, 0.33:Other, 1.00:Unknown
Predict
The client will subscribe a term deposit:
1 = Yes
0 = No


https://predict-bank-term-deposit-subscription.onrender.com/predict
PREDICTION FOR CUSTOMER INTEREST IN DEPOSITE TERM
Note: All variables are normalised.Please enter normalised values(Range: 0-1)
Age: Enter your Age
Job: Select option
List of Job:
0.00:Admin, 0.091:Blue-Collar, 0.182:Entrepreneur, 0.273:Housemaid, 0.364:Mangement, 0.454:Retired,
0.545:Self-employed,0.636:Services, 0.727:student, 0.818:Technician, 0.91:Unemployed, 1.00:Un-known
Marital Status: Select option
Marital Status: 0.0:Divorced/widow, 0.5:Married, 1.0:Single
Education: Select option
Level of Education: 0.0:Primary, 0.33:Secondary, 0.67:Tertiary, 1.0:Unknown
Credit Default: 0.0=No / 1.0=Yes
Balance: Enter Balance
Housing Loan: 0.0=No / 1.0=Yes
Personal Loan: 0.0=No / 1.0=Yes
Contact Communicattion Type: Select option
Contact Communicattion Types: 0.0:Cellular, 0.5:Telephone, 1.0:Unknown
Day: Enter workday number
Month: Enter month number
Months:
0.364:Jan, 0.273:Feb, 0.636:Mar, 0.00:April, 0.727:May, 0.545:jun, 0.454:Jul, 0.091:Aug, 1.00:Sep, 0.91:Oct, 0.818:Nov, 0.182:Dec
Campaign: Enter campaign days
Days Passed by: Enter number of days
Previous: Enter number of days
Previous Marketing Outcome: Select option
Previous Marketing Outcome: 0.00:Failure, 0.667:Success, 0.33:Other, 1.00:Unknown
Predict
The client will subscribe a term deposit: 1
1 = Yes
0 = No

16. References

This dataset is publicly available for research. The details are described in [Moro et al., 2011].

[Moro et al., 2011] S. Moro, R. Laureano and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In P. Novais et al. (Eds.), Proceedings of the European Simulation and Modelling Conference - ESM'2011, pp. 117-121, Guimaraes, Portugal, October 2011. EUROSIS.
Available at:
[pdf] http://hdl.handle.net/1822/14838

[bib] http://www3.dsi.uminho.pt/pcortez/bib/2011-esm-1.txt

Source:
Created by: Paulo Cortez (Univ. Minho) and Sergio Moro (ISCTE-IUL) @ 2012