Data Glacier

Your Deep Learning Partner

Bank Marketing(Campaign)

Prediction for Term Deposit subscription

12th May 2023

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

Background – Bank Marketing Campaign

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).

Approach

The analysis has been divided into following parts:

- Data Understanding
- Exploratory Data Analysis
- Univariate Analysis
- Correlation Analysis
- Bivariate Analysis
- Feature Engineering
- Model Building
- Model Evaluation
- Model Selection
- Model Deployment

Data Understanding

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.

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')
```

Hira Fahim

Exploratory Data Analysis

Step:1 Drop Duplicate Rows

```
# Remove duplicate rows
d=d.drop_duplicates()
d
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	5	184	3	success	yes
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0	unknown	no
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	2	188	11	other	no

45211 rows × 17 columns

Step:2 Drop Unnecessary Column

```
# 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 be discarded for a realistic predictive model.
d= d.drop(['duration'], axis=1)
```

```
d.shape
```

```
(45211, 16)
```

Step:3 Change Datatype of Categorical features

```
# 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")
```

Final Dataset

```
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
```

Hira Fahim

Univariate Analysis

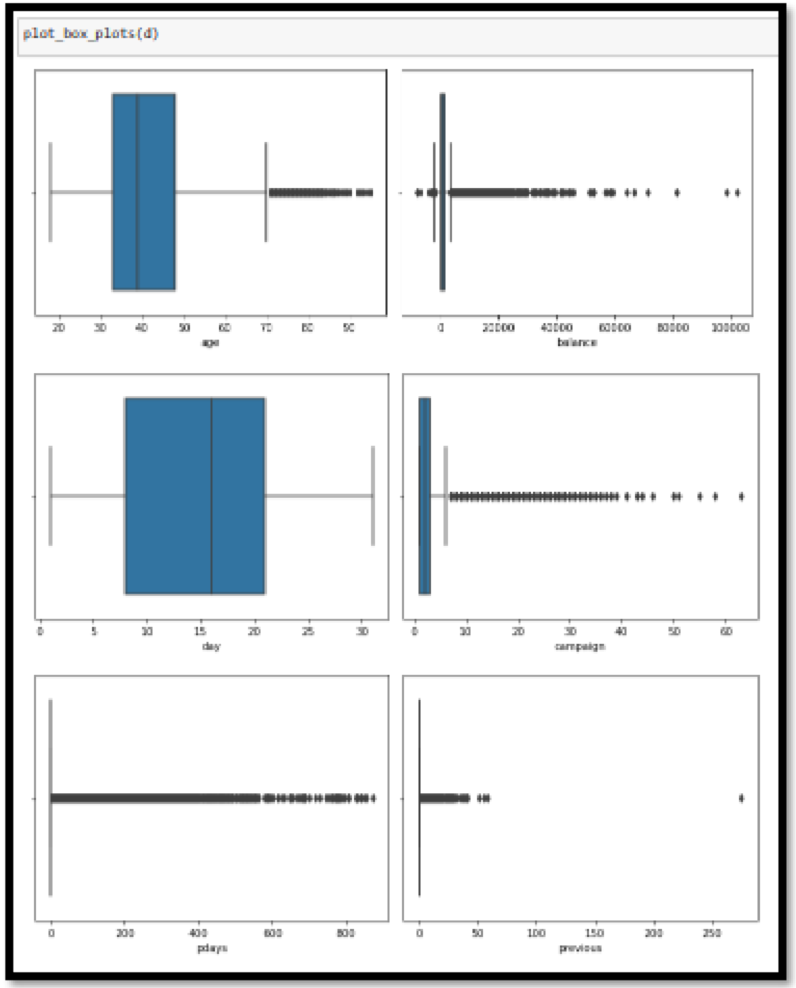
Description of the data

```
# 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

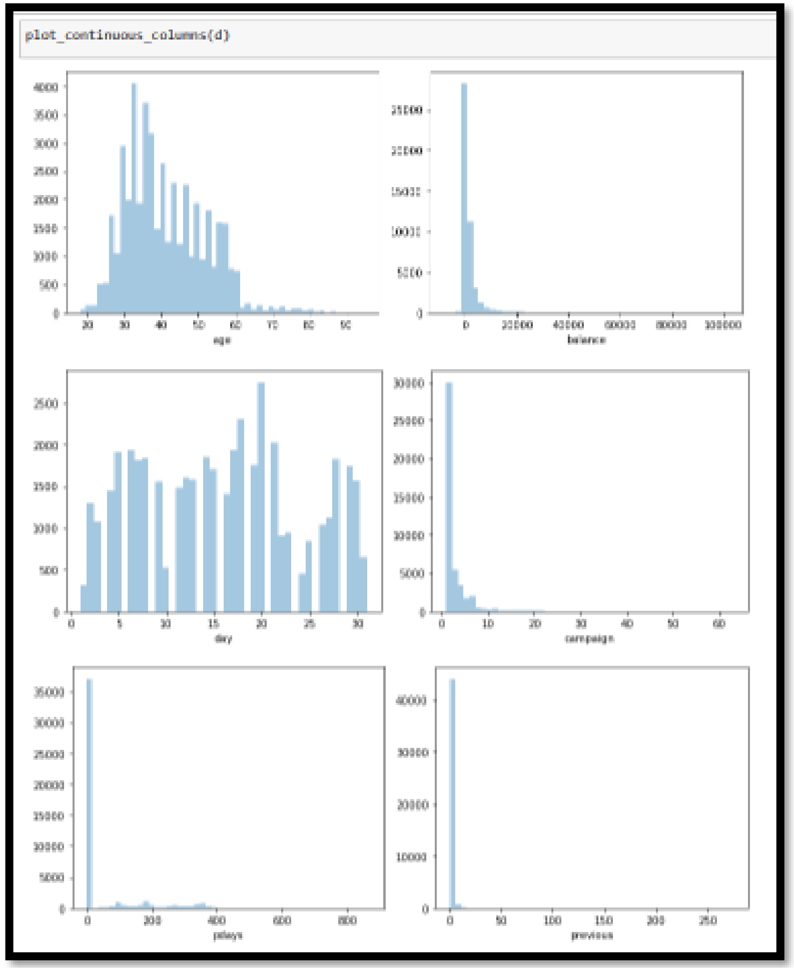
From **description** and **boxplot**, we can see there are outliers in numerical input variables like age, balance, campaign, pdays and previous. Pdays have most outliers comparatively.

Visualization (boxplot) of Numerical Attributes



Univariate Analysis

Histogram for Numerical Attributes

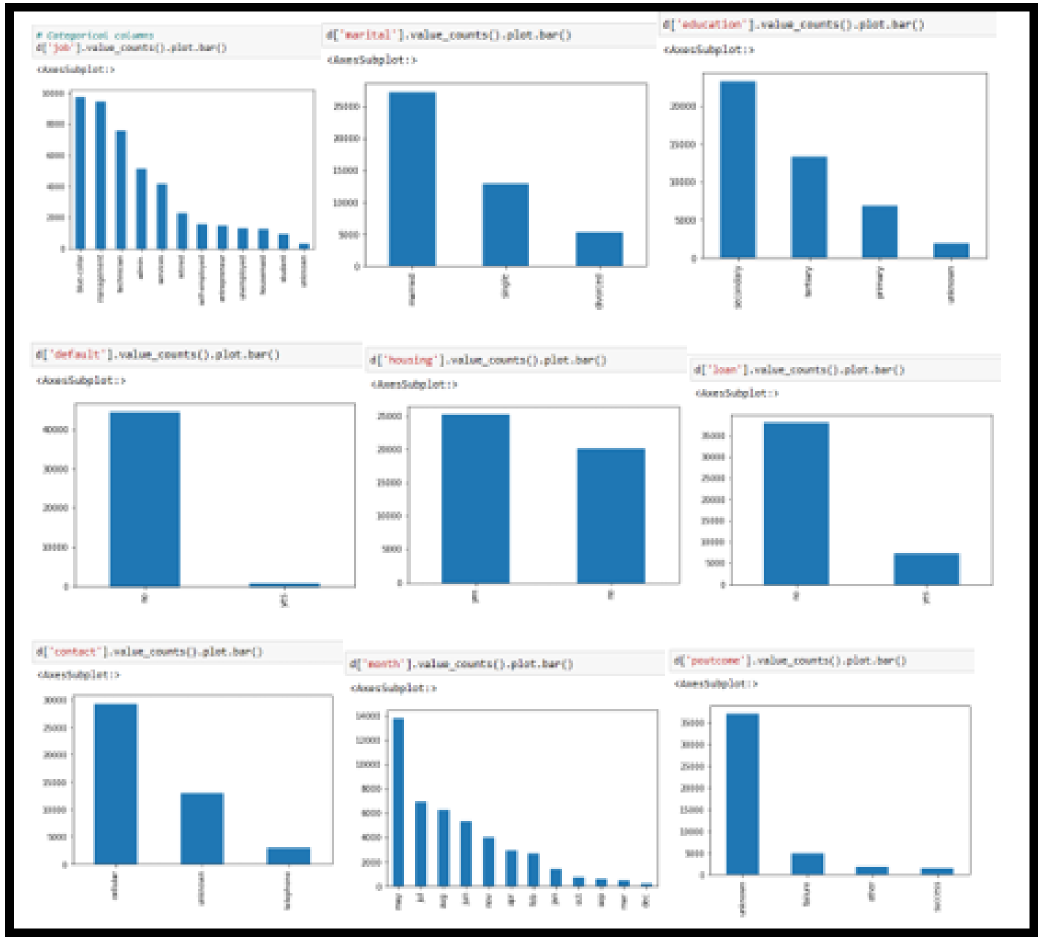


In Histogram, we can see input variables like age, balance, campaign, pdays and previous are **positively skewed**, and we can also see uneven distribution of data in day column.

Hira Fahim

Univariate Analysis

Visualization of Categorical Attributes



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

Correlation Analysis

Correlation between numerical input and Output variables

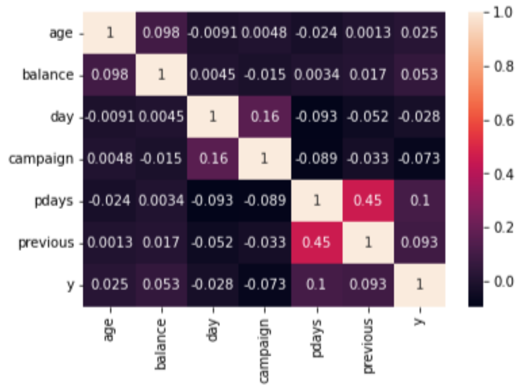
```
d.corr()
```

	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

Heat Map

```
sns.heatmap(d.corr(),annot=True)
```

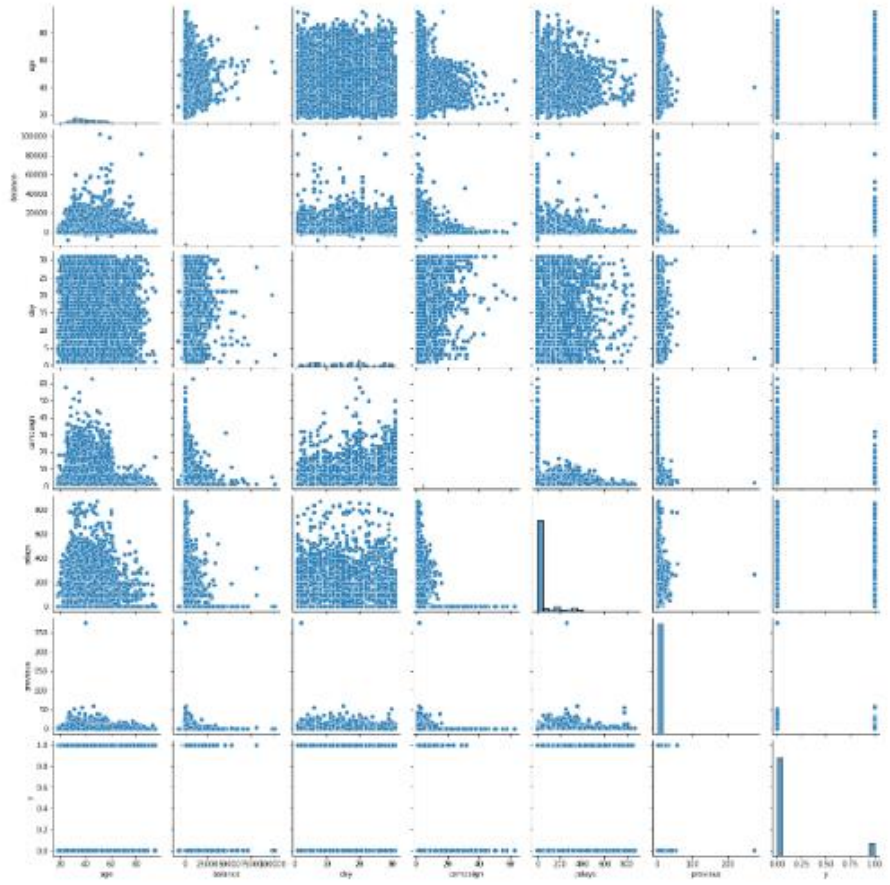
<AxesSubplot:>



Pairplot

```
sns.pairplot(d) # checking the relationship
```

<seaborn.axisgrid.PairGrid at 0x1d2e45934c8>



From Correlation Analysis, we see there is less correlation between numerical attribute and output variable

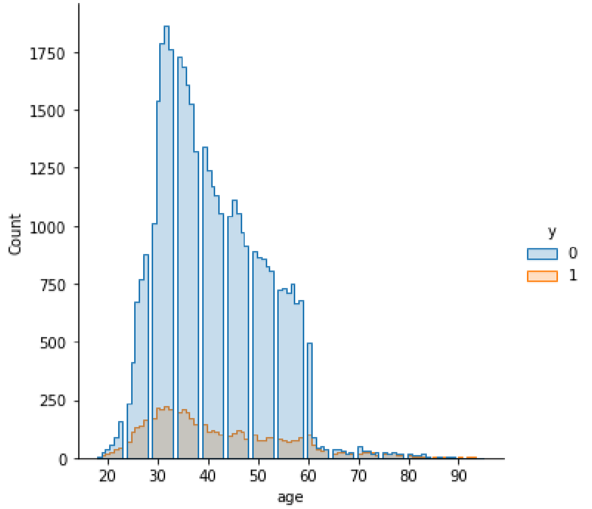
Hira Fahim

Bivariate Analysis

Term Deposit based on Age

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

<seaborn.axisgrid.FacetGrid at 0x1d2f0f0cfd0>

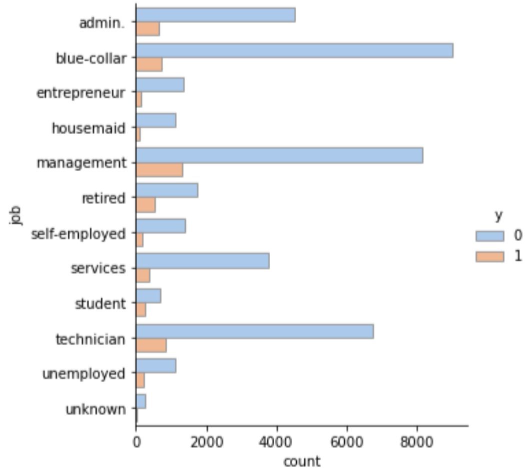


Clients between age 30-40 are more responsive towards term deposit

Term Deposit based on Job

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

<seaborn.axisgrid.FacetGrid at 0x1d2f108e670>

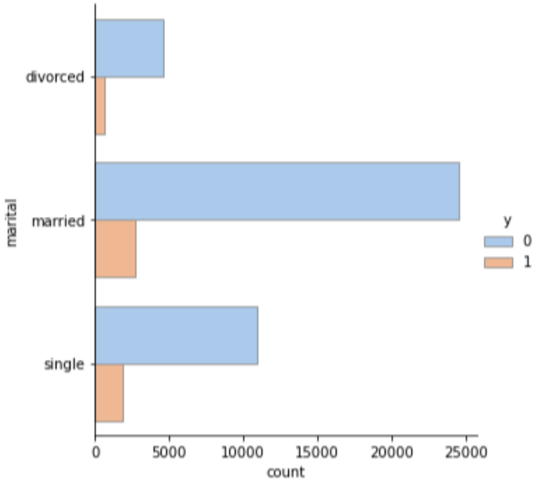


Clients with job related to 'management, blue-collar and technician' have subscribed for deposit.

Term Deposit based on Marital status

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

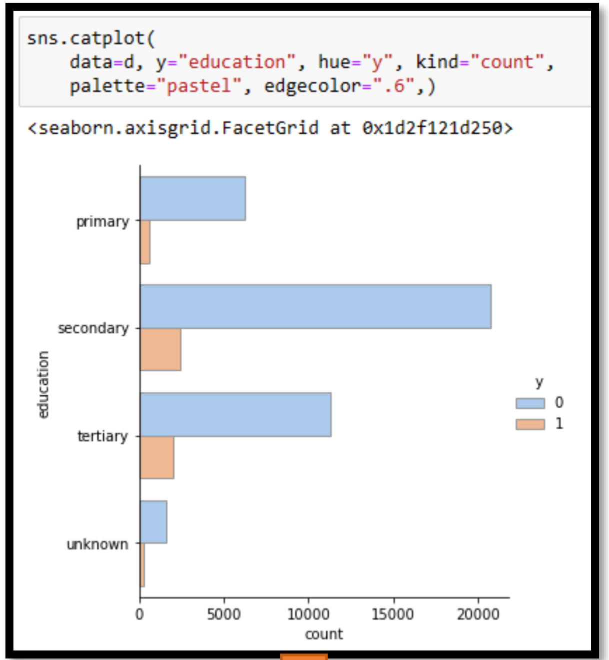
<seaborn.axisgrid.FacetGrid at 0x1d2f0edd9a0>



Married Clients are main contributor of deposit scheme.

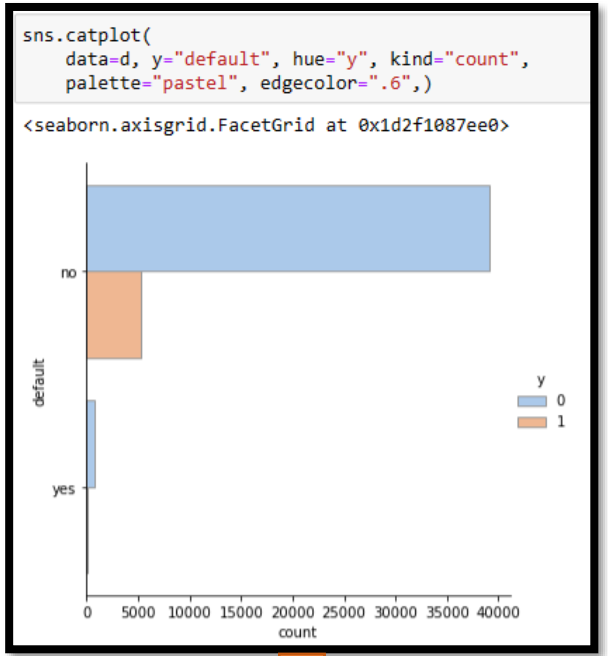
Bivariate Analysis

Term Deposit based on Education



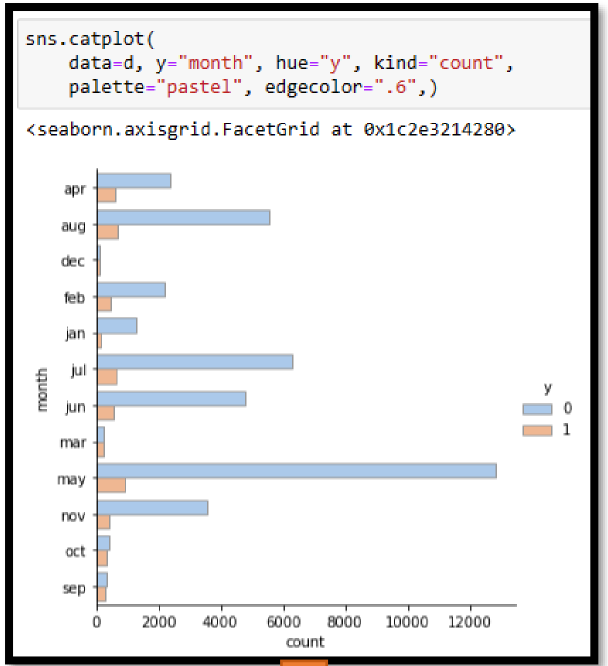
Clients with secondary and tertiary educational background are main contributors.

Term Deposit based on Credit default



Clients with good credit history are more interested in term deposit

Term Deposit based on Month

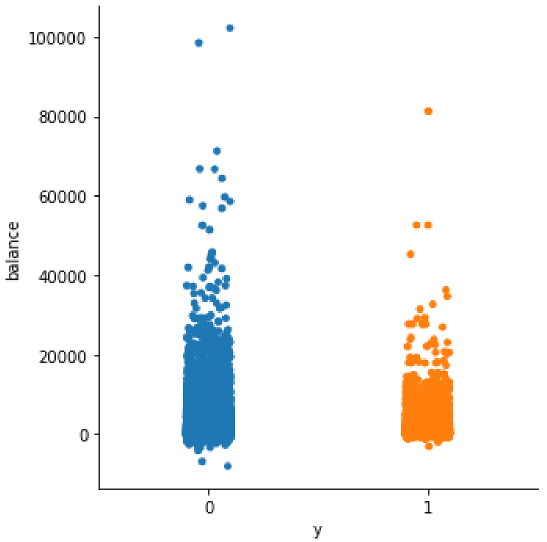


Investment in term deposit is fluctuating throughout the year but in the month of ‘May’ we have highest success rate.

Bivariate Analysis

Term Deposit based on Balance

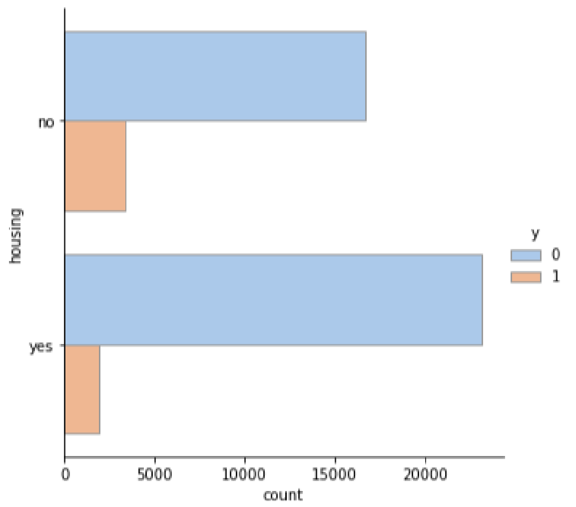
```
sns.catplot(data=d, x="y", y="balance")
<seaborn.axisgrid.FacetGrid at 0x1d2f1a36af0>
```



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

Term Deposit based on Housing Loan

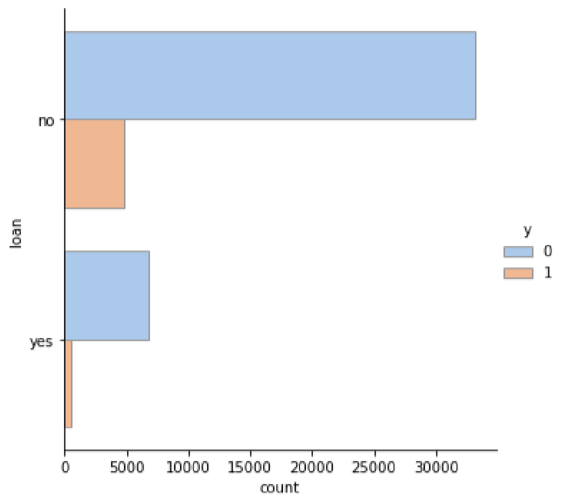
```
sns.catplot(
    data=d, y="housing", hue="y", kind="count",
    palette="pastel", edgecolor=".6",)
<seaborn.axisgrid.FacetGrid at 0x1d2f1c55f70>
```



Client with no housing loan 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>
```



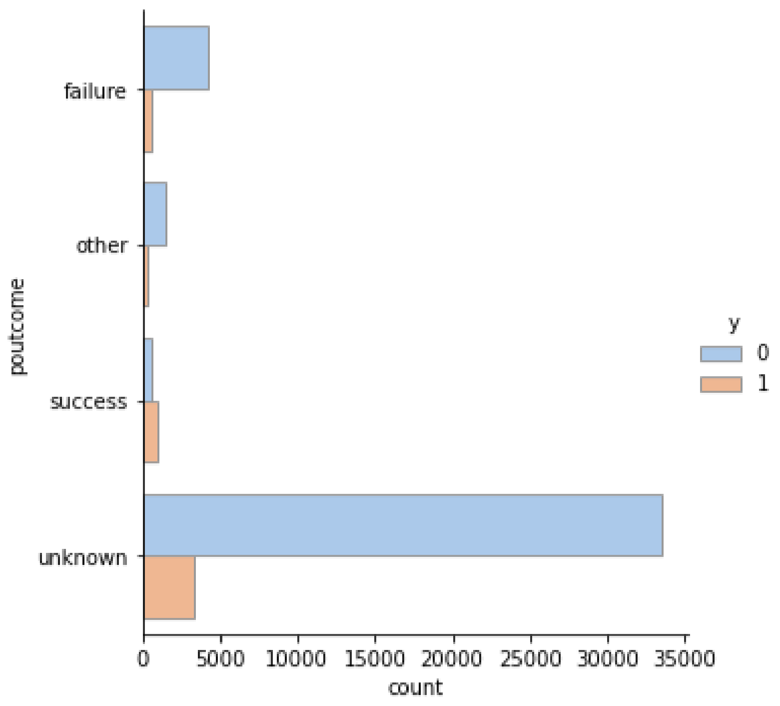
Client with no personal loan are more interested in term deposit.

Bivariate Analysis

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.

Feature Engineering

Step 1: Change datatype of Categorical Features

```
# 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")
```

Step 2: Encode Categorical Columns into Numerical using Label encoding

```
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'])
```

Feature Engineering

Last step: Feature Scaling using Normalization technique

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.  ]]
```

Final Dataset for Model building

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

Hira Fahim

Model Building

1. The dataset is imbalance, so we will balance the dataset using **SMOTE**.
2. 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).**
3. Different Machine Learning models to predict the term deposit subscription :
   - Logistic Regression
   - Random Forest
   - Gradient Boosting

Model Evaluation

Metrics of Evaluation

1. Accuracy, Precision, Recall and F1-Score
2. Scores of Test, Train and Complete dataset
3. Confusion Matrix
4. Lift and Gain
5. KS Statistics and ROC-AUC Score

Model Selection

Model selection based on Scores and Confusion Matrix

Model	Score All Dataset	Score Train Dataset	Score Test Dataset	TN	FP	FN	TP
Logistic Regression	0.63	0.66	0.66	7288	3572	4501	8593
Random Forest	0.76	0.72	0.71	9245	2544	4207	7958
Gradient Boosting	0.898	0.94	0.93	11442	347	1242	10923
Gradient Boosting + Hyperparameter Tuning	0.895	0.93	0.92	11385	404	1276	10886

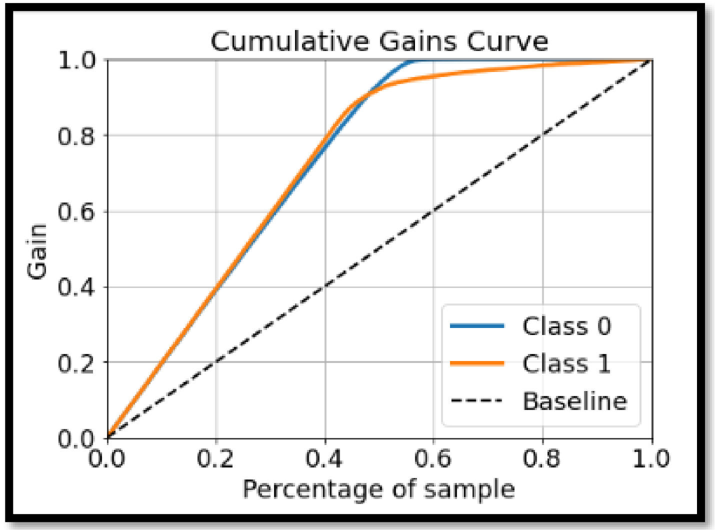
Based on scores and confusion matrix Gradient Boosting without hyperparameter tuning is the best model.

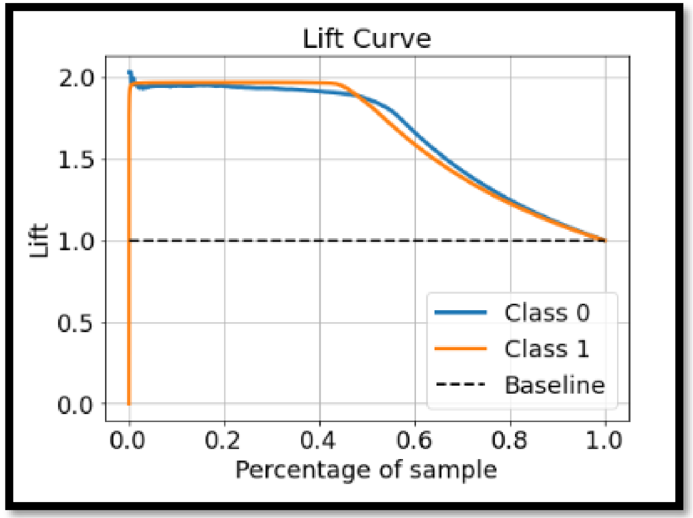
Model Selection

Model selection based on Lift and Gain Curve

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.

Gradient Boosting Classification Model





Model Selection

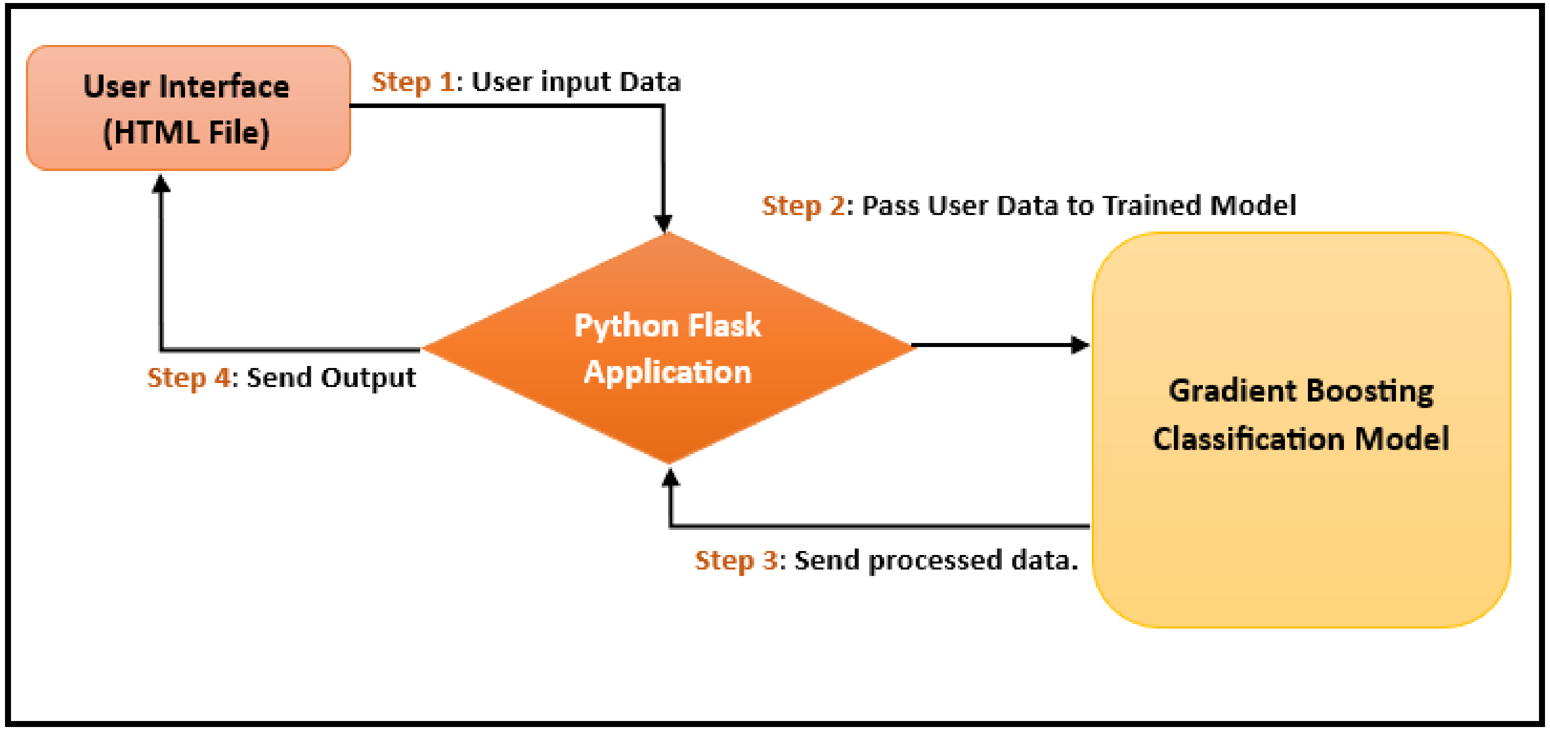
Model selection based on Accuracy, Precision, Recall, F1 Score, KS statistics and AUC-ROC

Model	Precision	Recall	F1 score	Accuracy	KS Statics	AUC- ROC
Logistic Regression	0 – 0.67 1 – 0.66	0.62 0.71	0.64 0.68	0.66	0.336	0.7246
Random Forest	0 –0.69 1 – 0.76	0.78 0.65	0.73 0.70	0.72	0.4475	0.785
Gradient Boosting	0 – 0.90 1 – 0.97	0.97 0.90	0.94 0.93	0.93	0.8697	0.97
Gradient Boosting + Hyperparameter Tuning	0 – 0.90 1 – 0.96	0.97 0.89	0.93 0.93	0.93	0.861	0.961

The F1- Score, KS statistics and AUC-ROC metrics of KNNC model are the best. Therefore, **The best model for deployment is Gradient Boosting Classification model.**

Model Deployment

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



Model Deployment

1. Save the model using Pickle.
2. Deploy the model using Flask framework.
3. 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.
4. 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.
5. The URL generate by 'app.py.' Open a web browser and navigate to http://127.0.0.1:5000/ following is output of Index.html.

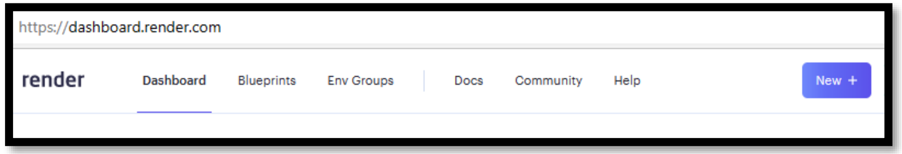
```python
import numpy as np
from flask import Flask, request, render_template
import pickle

#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()
```

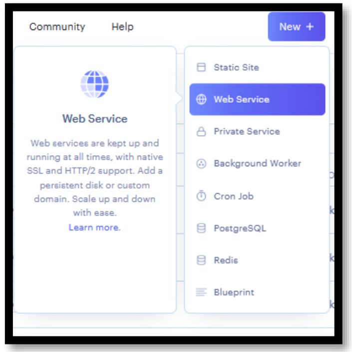
Model Deployment

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.

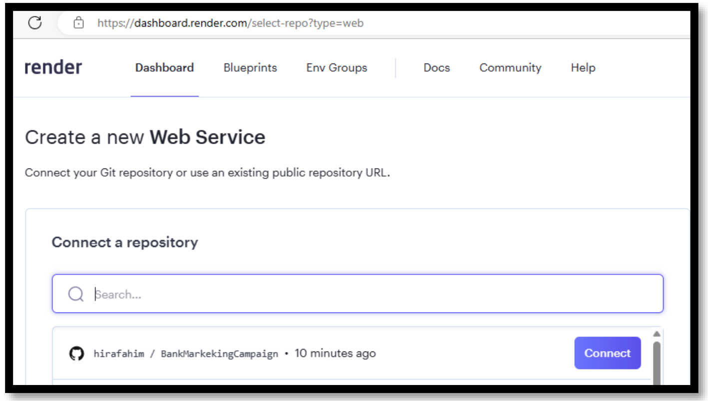
Model Deployment on Render (Open-Source Cloud Deployment)

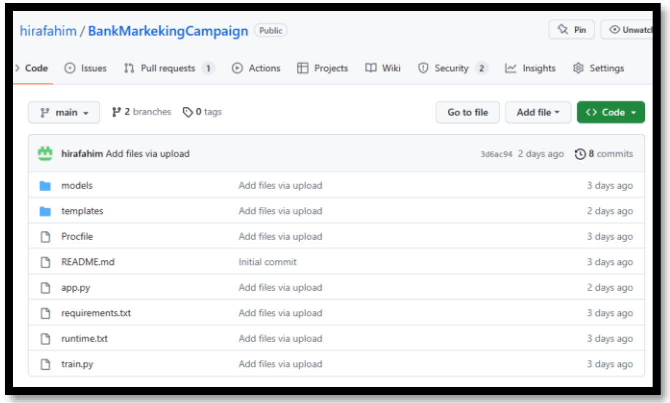
➢ After the model has been trained and deployed locally, now it is ready for deploy on open-source cloud "Render".

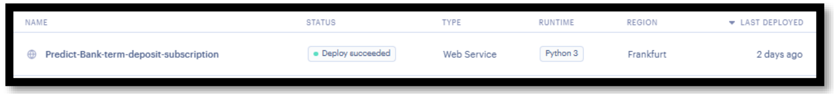




➢ Connect web service to GitHub Repository.



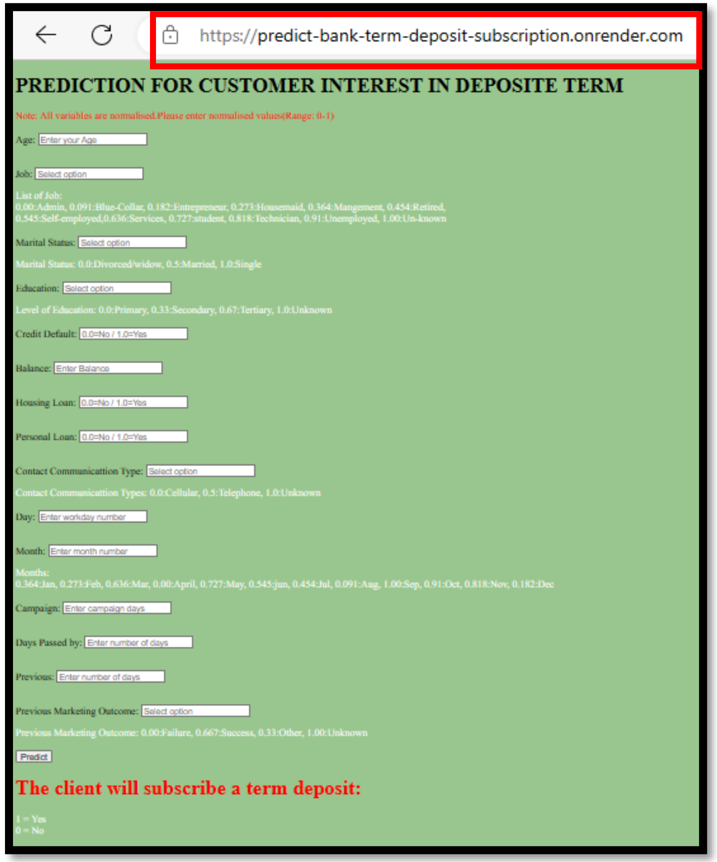


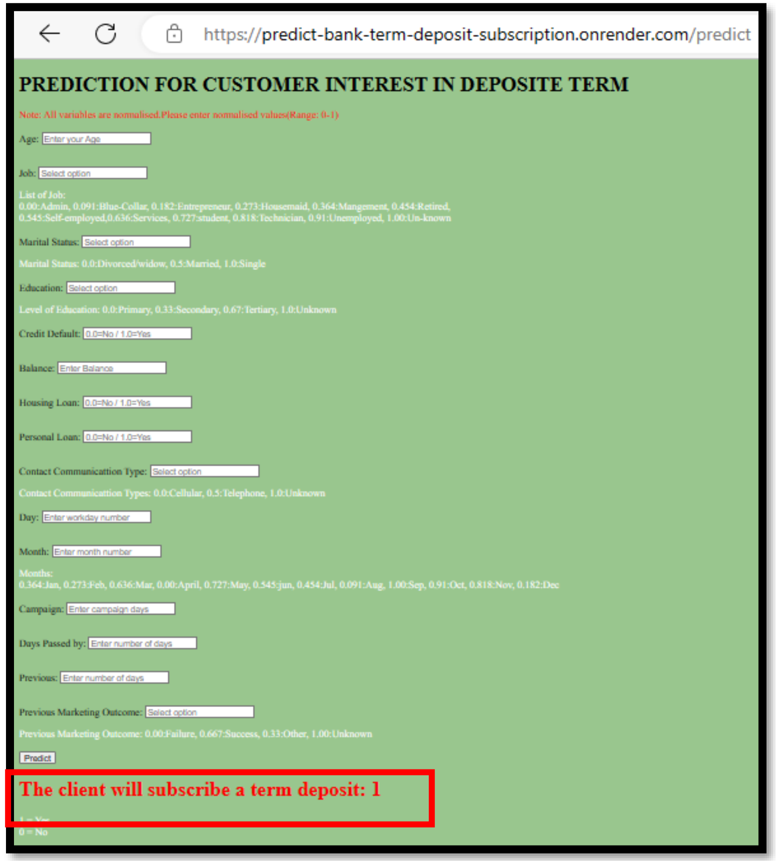


➢ Click and open the application for Term deposit subscription.

https://predict-bank-term-depositsubscription.onrender.com

Model Deployment on Render (Open-Source Cloud Deployment)





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.

Challenges

- Feature scaling was a challenging task, which is done normalization technique as the dataset consists of both numerical and categorical features.
- Selection of best model was also tricky but after carefully considering all parameters and metrics of evaluation choose ‘Gradient Boosting Classification model’ as the best model.

Thank You

Data Glacier
Your Deep Learning Partner

Hira Fahim