Suggestions for Marketing Campaign by a Bank

Group 2

B.Sc.(Hons.) Data Science

TEAM MEMBERS:

Akanksha Vasudeva Hanumapally

A Yasaswini

A Srinidhi

Hariharan Reddy

Bhavan's Vivekananda College
Sainikpuri
Autonomous College

CONTENT

AIM: To predict if a particular client has subscribed to the term deposit or not by building ML Algorithms

PATH TAKEN: The Data was first cleaned and dummified and then various classification and ensemble models were applied to it

PROBLEM STATEMENT

The data is related to 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,in order to access if the product bank term deposit would be subscribed ('yes') or ('not') subscribed.

*Refer to Appendix 1 for more info

ATTRIBUTE INFORMATION

Pdays: number of days that passed by after the client was last contacted from a previous campaign

Previous: number of contacts performed before this campaign and for this client

Poutcome: outcome of the previous marketing campaign

NUMERIC	
Age	Campaign
Pdays	Poutcome
Previous	

CATEGORICAL	
Job	Marital
Education	Default
Housing	Loan
Contact	Month
Day of week	Y

Target Variable

*Refer to Appendix 1 for more info

Data Preprocessing and Cleaning

Points to Note

-8 duplicate observations dropped.

-'unknown' label was found under,'job','marital','education','default','housing','loan'

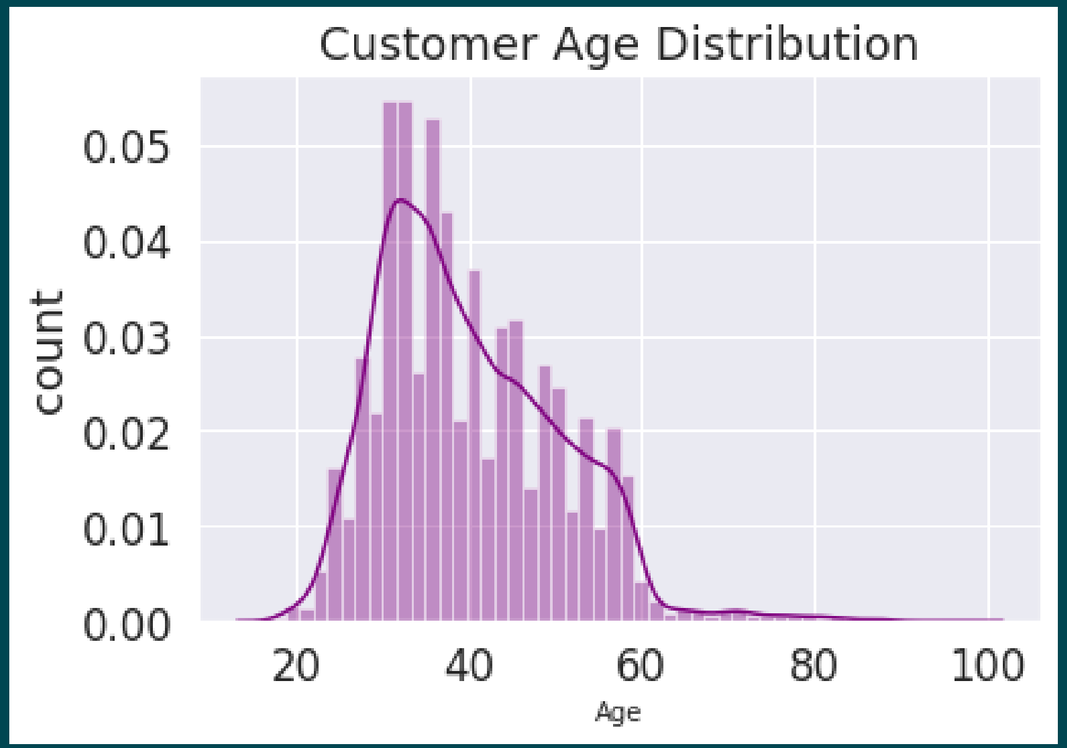
-The campaign is targeted towards Adults(17+)

Key Changes Made:

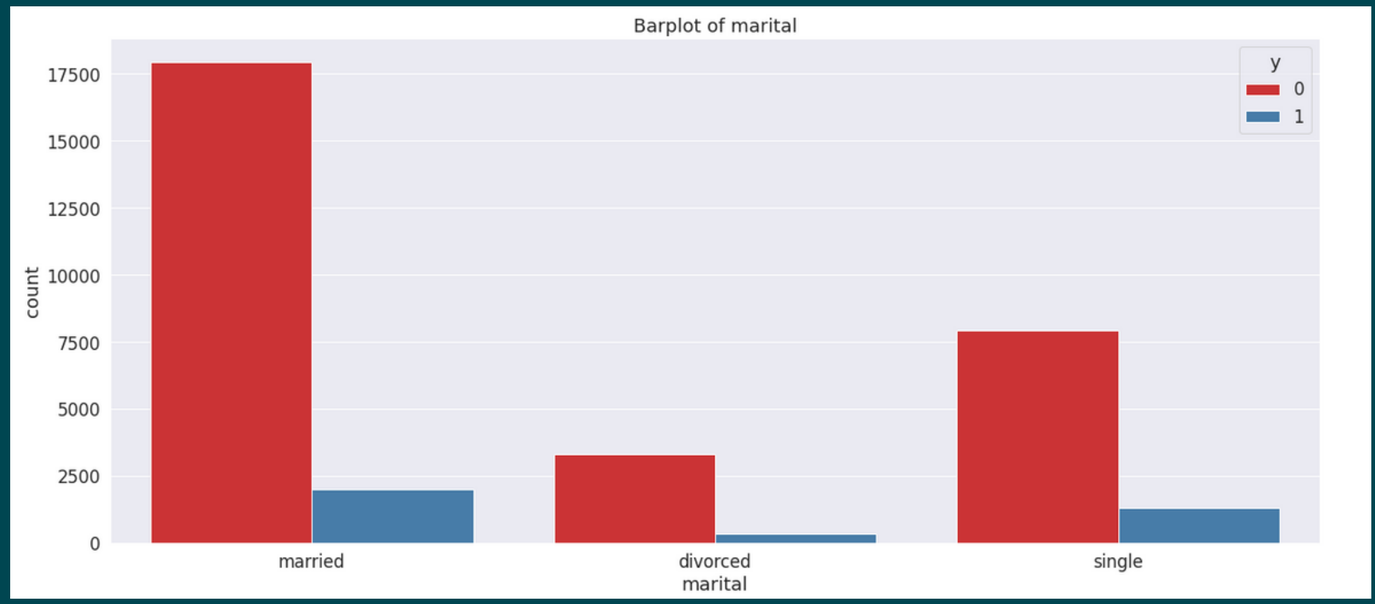
1. No missing values were found
2. Duplicate values were dropped.
3. Marital's 'unknown' label was dropped.
4. Outliers in the Campaign and Duration Variable were removed
5. The categorical features were dummified

*Refer to Appendix 2 for code

EXPLORATORY DATA ANALYSIS



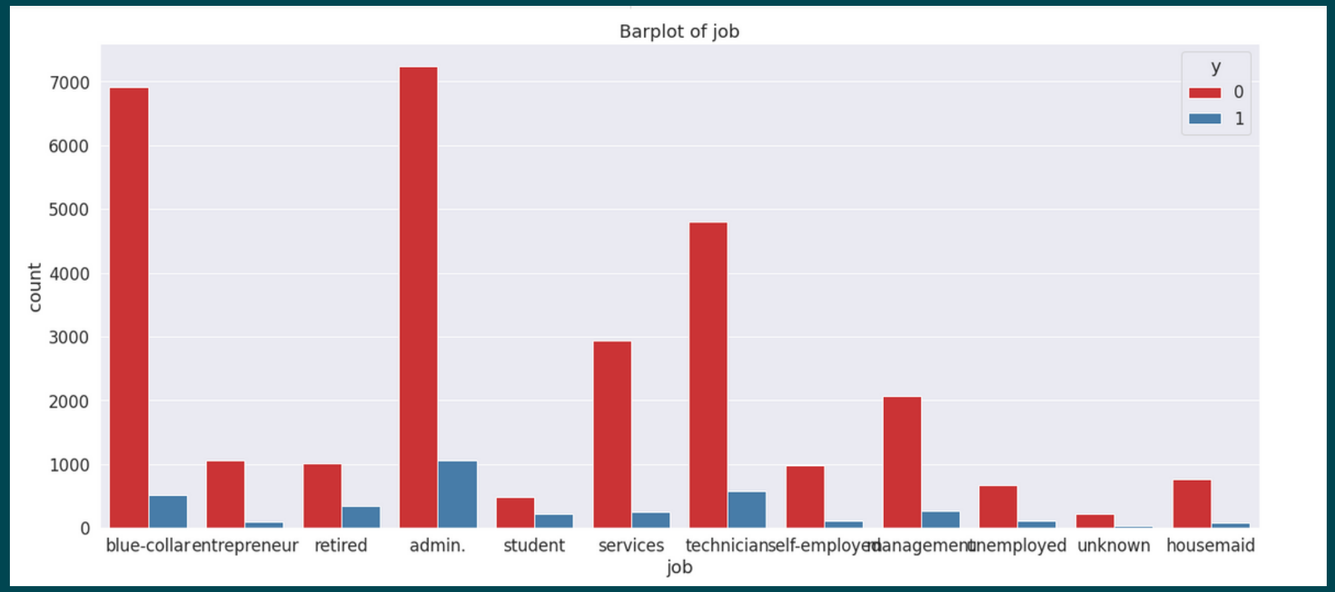
Majority of bank clients are married in status followed by single, divorced.



Clients between the age of 25 to 40 have subscribed to the deposit the most

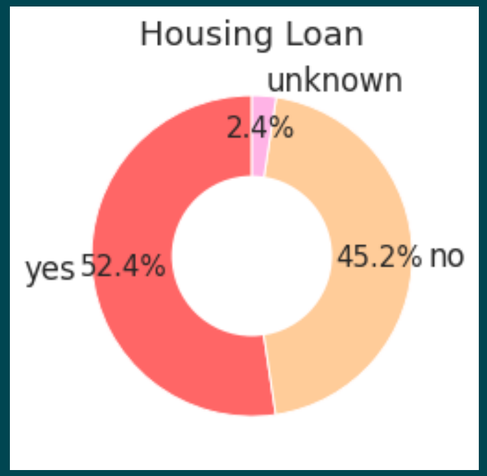
Not subscribed to Term Deposit

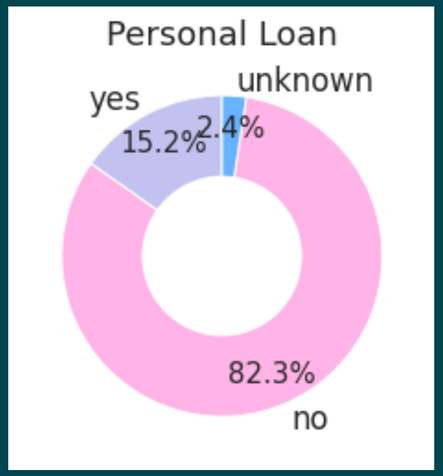
Subscribed to Term Deposit

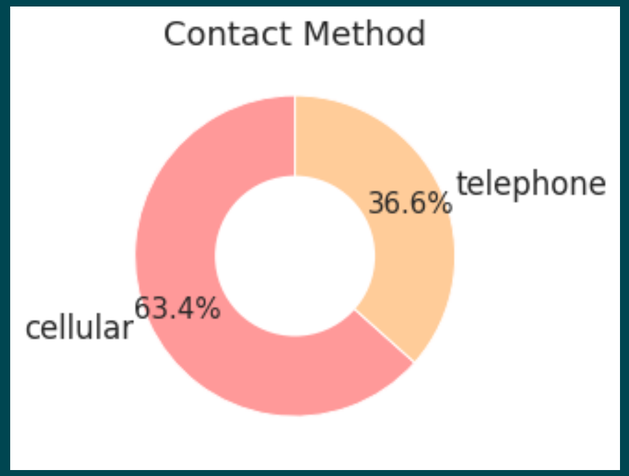


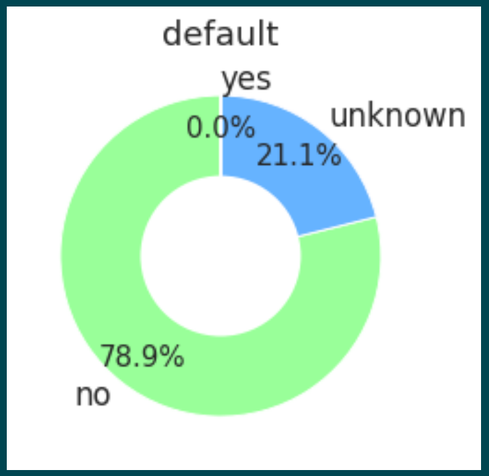
we can see client whose job profession is admin has subscribed more to the term deposit, but they are also highest when it comes to not subscribing.

Most of the clients have a housing loan and most of the clients were contacted through cellular means

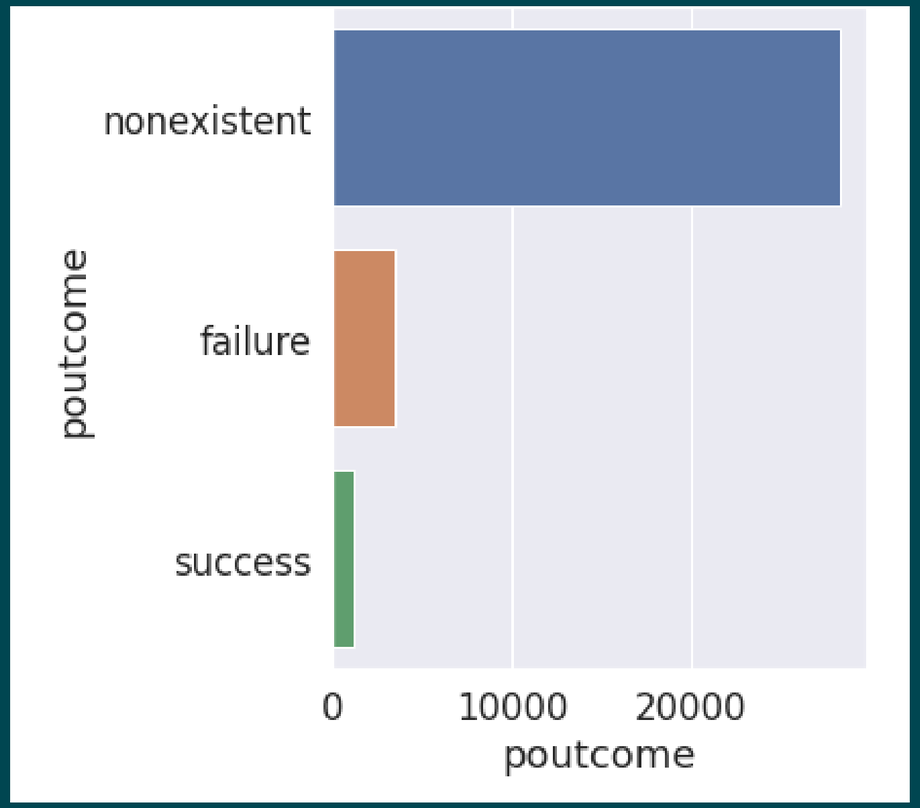




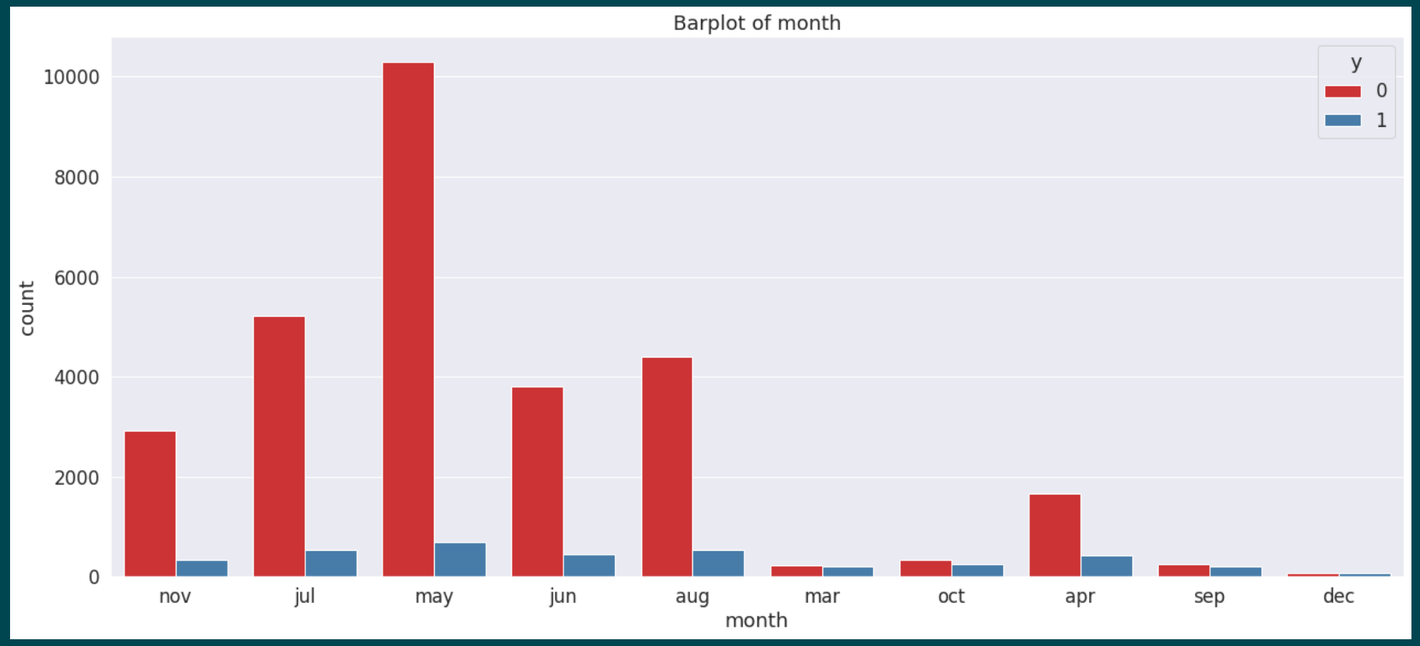




*Refer to Appendix 2 for code, more plots and detailed explanation



The results of most of the marketing campaigns were not known and the remaining data showed that the success rate was very low



The clients mostly subscribed to the term deposit during the months from May to August, as more communication took place in these months

Insights from EDA

Conclusion

Education, job, marital status are very important features for predicting whether the customer will subcribe to term deposit.

People who have university degree are showing more interest.

May-August are the month where client were contacted more by bank. Also customer are accepting more term deposits during this time.

Cellular phone is used more to communicate with the clients.

Most of the marketing campaign have mostly not made any difference and the success rate is very low

Logistic Regression

Different Train-Test Ratios Tried
80:20,70:30 and 60:40

Parameters for Comparison:
- Accuracy Score
- Confusion Matrix

Sno.	Train-Test Ratios	All Columns retained	Loan and Campaign Dropped	Loan, Campaign, Housing, Marital and Day of week dropped
1	70:30	90.81%	90.97%	90.90%
2	60:40	90.82	90.94	90.86
3	70:30	90.78	90.90	80.83

*Refer to Appendix 3 for code

Neural Networks

Different Train-Test Ratios Tried
80:20
70:30
60:40

GOOD

Different Architectures Tried
10-7-5-1
45-2-1
40-5-1
35-5-5-1
11-10-4-4-3-7-7-1

BAD

Optimizers Applied: Adam and SGD
Parameter of Comparison:
Accuracy Score and Loss

Top Models

No.	Train:Test	OPTIMIZER	ARCHITECTURE	EPOCHS	ACCURACY	LOSS
1	70:30	Adam	11-10-4-4-3-7-7-1	300	90.88%	0.2076
2	70:30	SGD	11-10-4-4-3-7-7-1	300	90.88%	0.2076
3	80:20	SGD	11-10-4-4-3-7-7-1	207	90.86%	0.2078
4	80:20	Adam	10-7-5-1	107	90.78%	0.2075

*Refer to Appendix 4 for code

Adam

Different Models 70:30

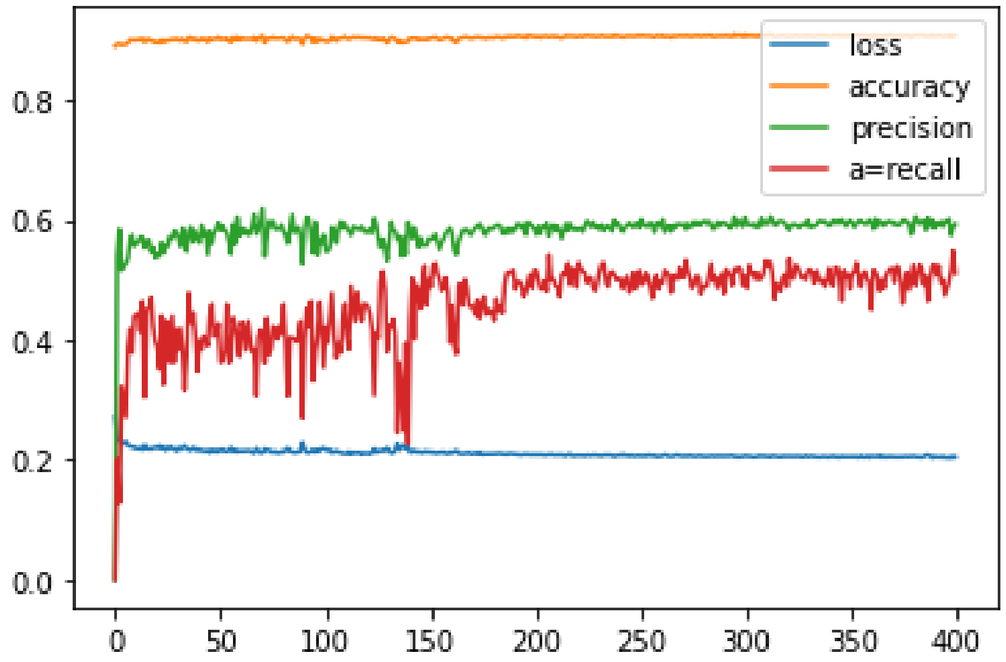
Optimizer	Architecture	Epochs	Accuracy	Loss
Adam	10-7-5-1	50	0.9056	0.2122
Adam	10-7-5-1	100	0.9057	0.2118
Adam	10-7-5-1	150	0.907	0.2092
Adam	10-7-5-1	200	0.9067	0.208
Adam	45-2-1	50	0.9049	0.2118
Adam	45-2-1	100	0.9058	0.2061
Adam	45-2-1	150	0.9064	0.2032
Adam	45-2-1	200	0.9053	0.2021
Adam	40-5-1	50	0.9019	0.2098
Adam	40-5-1	100	0.9058	0.2061
Adam	40-5-1	150	0.9064	0.2032
Adam	40-5-1	200	0.9053	0.2021
Adam	35-5-5-1	50	0.9023	0.2135
Adam	35-5-5-1	100	0.9073	0.2095
Adam	35-5-5-1	150	0.9072	0.2035
Adam	35-5-5-1	200	0.9076	0.2008
Adam	11-10-4-4-3-7-7-1	50	0.9045	0.2137
Adam	11-10-4-4-3-7-7-1	100	0.8997	0.2196
Adam	11-10-4-4-3-7-7-1	150	0.9029	0.213
Adam	11-10-4-4-3-7-7-1	200	0.9059	0.2087
Adam	11-10-4-4-3-7-7-1	250	0.9063	0.2079
Adam	11-10-4-4-3-7-7-1	300	0.9088	0.2076
Adam	11-10-4-4-3-7-7-1	350	0.9073	0.2058
Adam	11-10-4-4-3-7-7-1	400	0.9071	0.2057

Best Model

Architecture:11-10-4-4-3-7-7-1

Epochs: 300

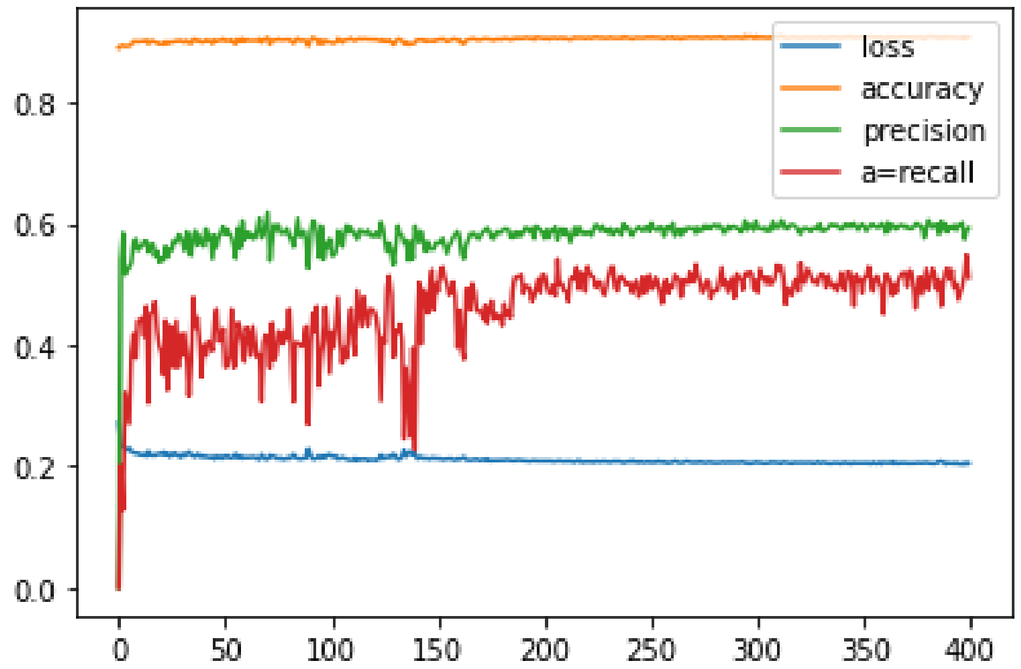
Accuracy:0.9088



Different Models 70:30

SGD

Best Model
Architecture:11-10-4-4-3-7-7-1
Epochs: 300
Accuracy:0.9088



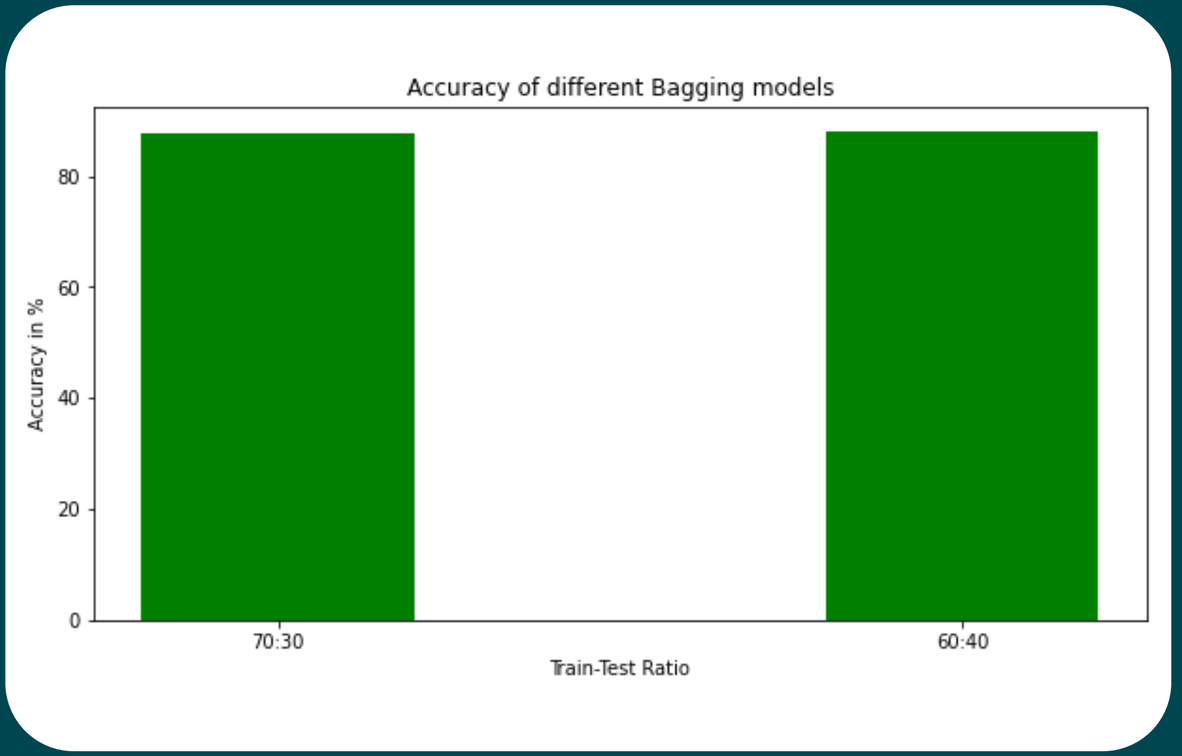
Optimizer	Architecture	Epochs	Accuracy	Loss
SGD	10-7-5-1	50	0.8894	0.3006
SGD	10-7-5-1	100	0.8894	0.2938
SGD	10-7-5-1	150	0.8908	0.2794
SGD	10-7-5-1	200	0.8927	0.2733
SGD	45-2-1	50	0.892	0.2801
SGD	45-2-1	100	0.8924	0.2773
SGD	45-2-1	150	0.8922	0.2691
SGD	45-2-1	200	0.8928	0.2635
SGD	40-5-1	50	0.8918	0.2809
SGD	40-5-1	100	0.8924	0.2742
SGD	40-5-1	150	0.8928	0.2661
SGD	40-5-1	200	0.892	0.2612
SGD	35-5-5-1	50	0.8922	0.2829
SGD	35-5-5-1	100	0.8912	0.2744
SGD	35-5-5-1	150	0.8925	0.2663
SGD	35-5-5-1	200	0.8921	0.2622
SGD	11-10-4-4-3-7-7-1	50	0.9045	0.2137
SGD	11-10-4-4-3-7-7-1	100	0.8997	0.2196
SGD	11-10-4-4-3-7-7-1	150	0.9029	0.213
SGD	11-10-4-4-3-7-7-1	200	0.9059	0.2087
SGD	11-10-4-4-3-7-7-1	250	0.9063	0.2079
SGD	11-10-4-4-3-7-7-1	300	0.9088	0.2076
SGD	11-10-4-4-3-7-7-1	350	0.9073	0.2058
SGD	11-10-4-4-3-7-7-1	400	0.9071	0.2057

Inferences from ANN

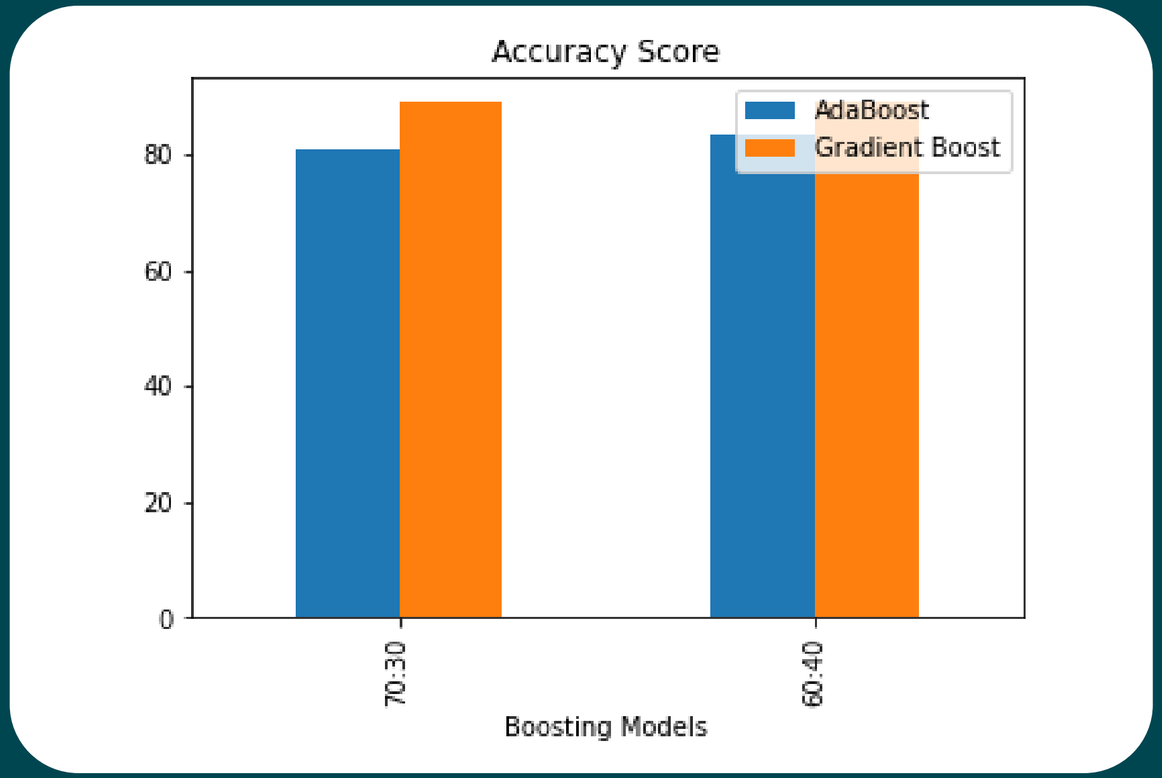
1. Train:Test= 70:30 performed the best
2. Higher Epochs values gave better results
3. There was no difference in the performance of both the optimizers
4. There is hardly any change in loss values for all ratios and architectures.

-Bagging-

Train-Test Ratio	Accuracy Score
70:30	88.08%
60:40	87.64%%



-Boosting-



Boosting Algorithm Applied	Train-Test Ratio	Accuracy Score
AdaBoost	70:30	81.13%
GradientBoost	70:30	88.89%
AdaBoost	60:40	83.63%
GradientBoost	60:40	88.93%

Train Test Ratio: 70:30

Other Algorithms Applied

Accuracy Scores

Decision Tree → **84.21%**

Random Forest → **89.81%**

SVM → **90.66%**

KNN → **89.77%**

Naive Bayes → **87%**

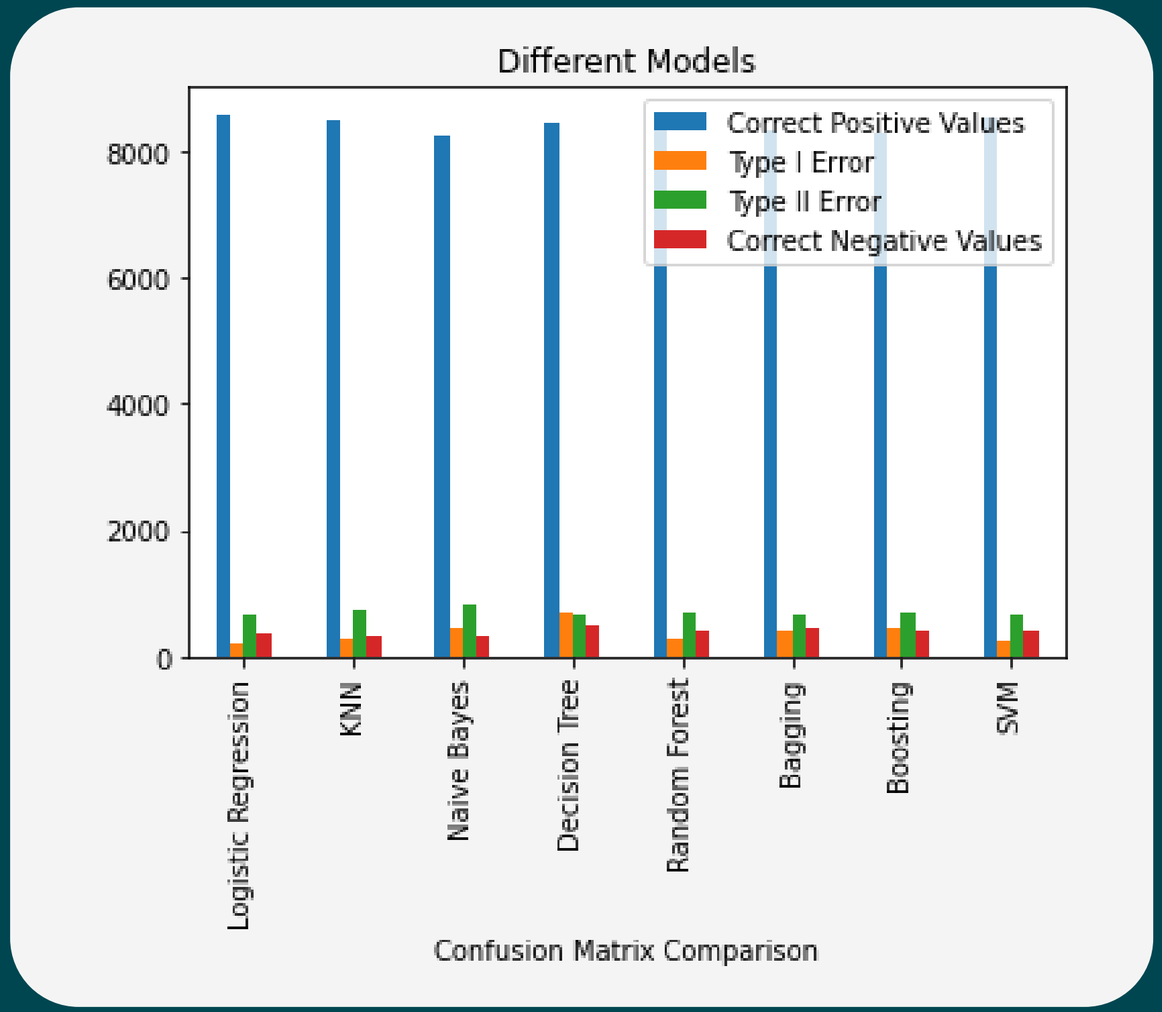
*Refer to Appendix 7 for code

Choosing Ratio for Comparison

70:30

We observe that in most of the cases 70:30 yields the most accurate results

CONFUSION MATRIX COMPARISON



Logistic Regression		KNN		Naive Bayes		Decision Tree	
8578	217	8497	307	8256	462	8446	686
680	389	743	317	846	321	651	481

Random Forest		Bagging		Boosting		SVM	
8446	307	8337	416	8290	463	8540	255
717	415	664	468	701	431	666	394

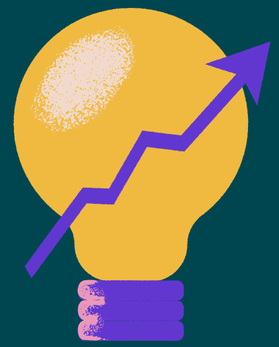
*Refer to Appendix 8 for more info

CONCLUSION

- In conclusion our top performing algorithms are Logistic Regression and Neural Networks
- Logistic Regression where Loan and Campaign are dropped
- It can be said that Neural Networks best fits our dataset with:
- Optimizers: SGD and Adam
- Architecture: 11-10-4-4-3-7-7-1
- Epochs:300
- Logistic Regression is not considered the best fit because of high Type 2 error values

Algorithm	Accuracy %
Logistic Regression	90.97%
Neural Networks	90.88%
SVM	90.66%
Random Forest	89.81%
KNN	89.77%
Boosting	88.89%
Bagging	87.86%
Naïve Bayes	86.76%
Decision Tree	84.21%

FUTURE SCOPE AND RECOMMENDATIONS



- This study is helpful in giving one insights about predictive models suitable for a bank's marketing campaign analysis.
- It was observed that most of the bank's campaigns failed, in order to perform better the bank should target the following people more:

1. Married Persons
2. University Graduates
3. Mostly the youth ranging between 20's to 40's

Thanks for your valuable time!

Akanksha Vasudeva Hanumapally
A Yasaswini
A Srinidhi
Hariharan Reddy

Click here the icons below

Appendix 1

Term deposit is a deposit that a bank or a financial instiurion offers with a fixed rate (often better than just opening deposit account) in which your money will be returned back at a specific maturity time.

PROBLEM STATEMENT

The data is related to 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,in order to access if the product bank term deposit would be subscribed ('yes') or ('not') subscribed.

AIM: To predict if a particular client has subscribed to the term deposit or not

PATH TAKEN: The Data was first cleaned and dummified and then various classification and ensemble models were applied to it

ATTRIBUTE INFORMATION

Feature	type	Description
age	numeric	age of a person
job	categorical	type of job
marital	categorical	marital status
education	categorical	education qualification of a customer
Default	categorical	has credit in default?
housing	categorical	has housing loan?
loan	categorical	has personal loan?
contact	categorical	contact communication type

Target Variable:

Y	**binary**	**has the client subscribed a term deposit?**

Feature	type	Description
month	categorical	last contact month of the year
day of week	categorical	last contact day of the week
duration	numeric	last contact duration,in seconds
campaign	numeric	number of contacts performed during this campaign and for this client
pdays	numeric	number of days that passed by after the client was last contacted from a previous campaign
previous	numeric	number of contacts performed before this campaign and for this client
poutcome	categorical	outcome of the previous marketing campaign

Appendix 2

Data Cleaning

2. Checking for different datatypes in the dataset

+ Code + Text

```
[8]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32950 entries, 0 to 32949
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          32950 non-null  int64 
 1   job          32950 non-null  object
 2   marital      32950 non-null  object
 3   education    32950 non-null  object
 4   default      32950 non-null  object
 5   housing      32950 non-null  object
 6   loan         32950 non-null  object
 7   contact      32950 non-null  object
 8   month        32950 non-null  object
 9   day_of_week  32950 non-null  object
 10  duration     32950 non-null  int64 
 11  campaign     32950 non-null  int64 
 12  pdays        32950 non-null  int64 
 13  previous     32950 non-null  int64 
 14  poutcome     32950 non-null  object
 15  y            32950 non-null  object
dtypes: int64(5), object(11)
memory usage: 4.0+ MB
```

It can be observed that the data has 10 categorical features and 5 numeric fearures. The output variable i.e. y is boolean in nature, here 'yes' means that the client has subscribed to the term deposit and vice-versa

Appendix 2

Data Cleaning

3. Checking for missing values

```
df.isnull().sum()
```

```
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
y               0
dtype: int64
```

It can be observed that our dataset has no null/missing values

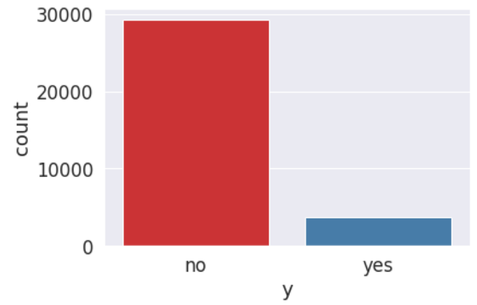
4. Checking how many clients have subscribned to the term deposit

```
[12] df.y.value_counts()
```

```
no     29238
yes     3712
Name: y, dtype: int64
```

It can be observed that 3712 clients have subscribed to the term deposit

```
[13] sns.set(font_scale=1.5)
countplt=sns.countplot(x='y', data=df, palette ='Set1')
plt.show()
```



The data is very unbalanced. The count of 'no' is much higher than 'yes'

Appendix 2

Data Cleaning

5. Checking for duplicate values

```
[14] df.duplicated().sum()
```

```
8
```

It can be observed that our dataset has 8 duplicate values, we now drop them

```
[15] df = df.drop_duplicates()
```

```
[16] df.y.value_counts()
```

```
no     29230
yes     3712
Name: y, dtype: int64
```

7. Describing Numerical Variables

```
[25] df.describe()
```

	age	duration	campaign	pdays	previous
count	32877.000000	32877.000000	32877.000000	32877.000000	32877.000000
mean	40.012167	258.022812	2.559571	962.091097	0.174499
std	10.401993	258.903818	2.749534	187.856729	0.498870
min	17.000000	0.000000	1.000000	0.000000	0.000000
25%	32.000000	103.000000	1.000000	999.000000	0.000000
50%	38.000000	180.000000	2.000000	999.000000	0.000000
75%	47.000000	319.000000	3.000000	999.000000	0.000000
max	98.000000	4918.000000	56.000000	999.000000	7.000000

Appendix 2

Data Cleaning

6. Dealing with data points under the label 'unknown'

```
print("# Missing value 'job' variable: {0}".format(len(df.loc[df['job'] == "unknown"])))
print("# Missing value 'marital' variable: {0}".format(len(df.loc[df['marital'] == "unknown"])))
print("# Missing value 'education' variable: {0}".format(len(df.loc[df['education'] == "unknown"])))
print("# Missing value 'default' variable: {0}".format(len(df.loc[df['default'] == "unknown"])))
print("# Missing value 'housing' variable: {0}".format(len(df.loc[df['housing'] == "unknown"])))
print("# Missing value 'loan' variable: {0}".format(len(df.loc[df['loan'] == "unknown"])))
print("# Missing value 'contact' variable: {0}".format(len(df.loc[df['contact'] == "unknown"])))
print("# Missing value 'month' variable: {0}".format(len(df.loc[df['month'] == "unknown"])))
print("# Missing value 'day_of_week' variable: {0}".format(len(df.loc[df['day_of_week'] == "unknown"])))
print("# Missing value 'poutcome' variable: {0}".format(len(df.loc[df['poutcome'] == "unknown"])))
```

```
# Missing value 'job' variable: 265
# Missing value 'marital' variable: 65
# Missing value 'education' variable: 1396
# Missing value 'default' variable: 6939
# Missing value 'housing' variable: 796
# Missing value 'loan' variable: 796
# Missing value 'contact' variable: 0
# Missing value 'month' variable: 0
# Missing value 'day_of_week' variable: 0
# Missing value 'poutcome' variable: 0
```

```
df[df['marital'] == "unknown"]
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
252	34	admin.	unknown	university.degree	no	yes	no	cellular	jul	thu	243	1	999	0	nonexistent	no
289	31	entrepreneur	unknown	university.degree	no	no	no	cellular	oct	thu	164	1	999	0	nonexistent	yes
956	50	unemployed	unknown	basic.9y	no	yes	yes	cellular	nov	thu	139	1	999	0	nonexistent	no
978	31	entrepreneur	unknown	university.degree	no	yes	no	telephone	oct	thu	157	4	999	1	failure	no
1112	49	unknown	unknown	unknown	unknown	yes	yes	cellular	jul	mon	49	9	999	0	nonexistent	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30306	45	unknown	unknown	unknown	no	yes	no	cellular	jul	wed	586	1	999	0	nonexistent	yes
30747	66	retired	unknown	basic.9y	no	yes	no	cellular	aug	wed	340	2	999	0	nonexistent	no
31362	66	retired	unknown	basic.9y	no	yes	no	cellular	aug	wed	810	3	999	2	failure	yes
32135	25	technician	unknown	university.degree	no	yes	yes	cellular	may	mon	150	1	999	0	nonexistent	no
32862	31	blue-collar	unknown	unknown	unknown	no	no	telephone	may	tue	170	2	999	0	nonexistent	no

65 rows × 16 columns

```
[21] df.drop(df[df['marital'] == "unknown"].index, inplace=True)
     df.shape
```

We retain the 'unknown' category in the attribute 'default' as they are high in number

```
[23] df.education.value_counts()
```

```
university.degree      9706
high.school            7584
basic.9y               4819
professional.course    4187
basic.4y               3318
basic.6y               1860
unknown                1387
illiterate               16
Name: education, dtype: int64
```

```
df.job.value_counts()
```

```
admin.           8301
blue-collar      7428
technician       5388
services         3190
management       2343
retired          1361
entrepreneur     1157
self-employed    1095
housemaid         852
unemployed        795
student           710
unknown           257
Name: job, dtype: int64
```

Key Points

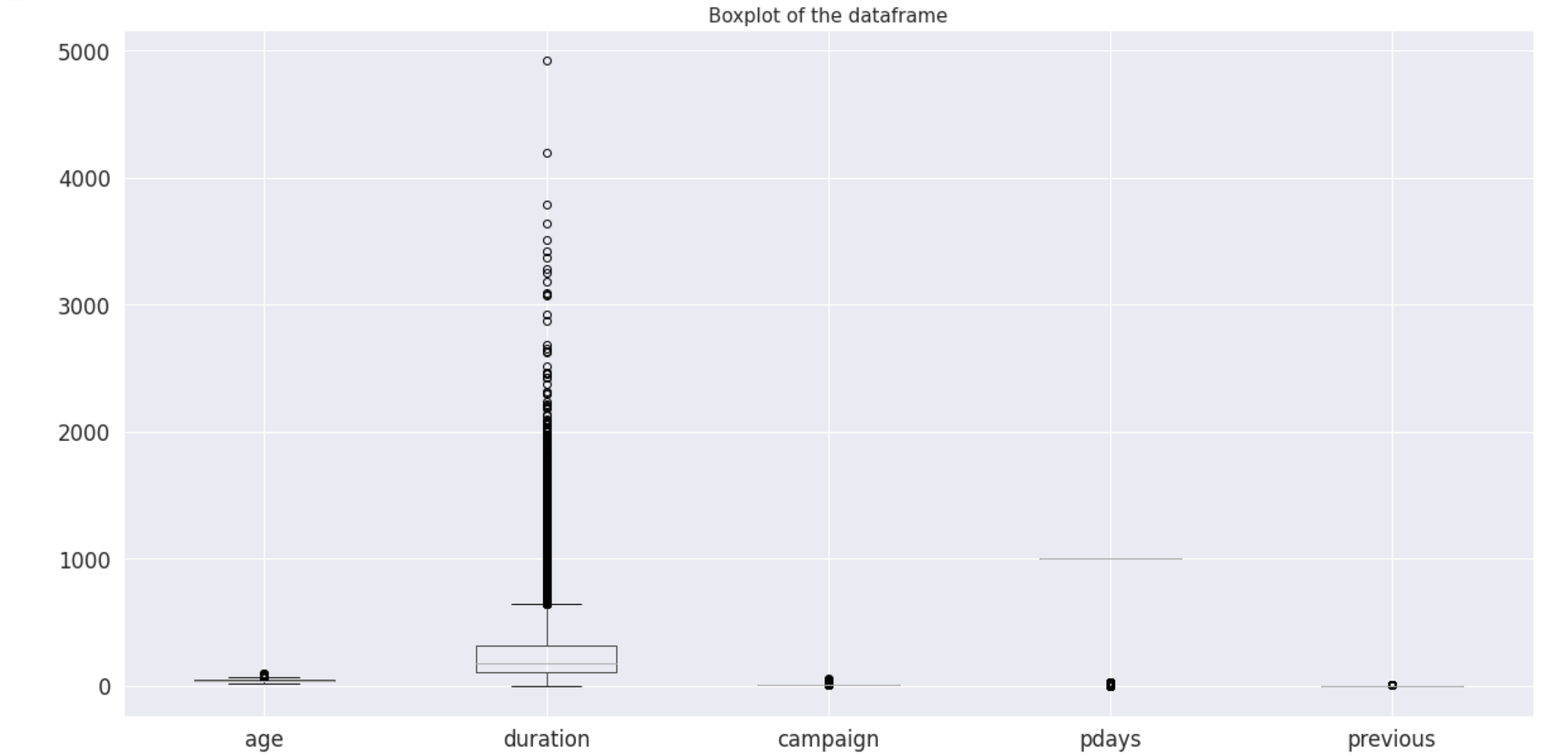
1. The minimum age here is 17, indicating that the campaign mainly targets adults
2. 'duration' indicates last contact duration(in seconds), if the duration value is '0', the y(Output variable) will always be 'no'

Appendix 2

EDA

1. Checking for outliers

```
plt.figure(figsize=(20,10))
df.boxplot()
plt.title("Boxplot of the dataframe", fontsize = 15)
print()
```



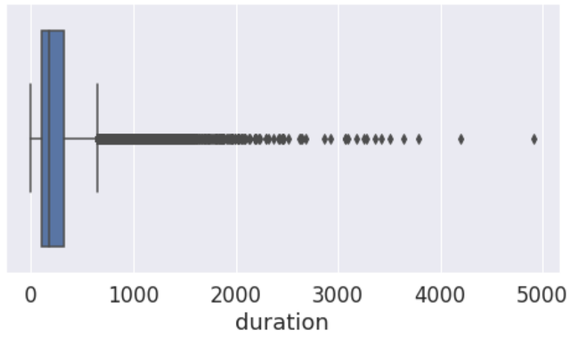
It can be ovserved that 'duration' has a high number of outliers. 'age' also has few outliers.
Now we individually plot these features to gain better understanding

Appendix 2

EDA

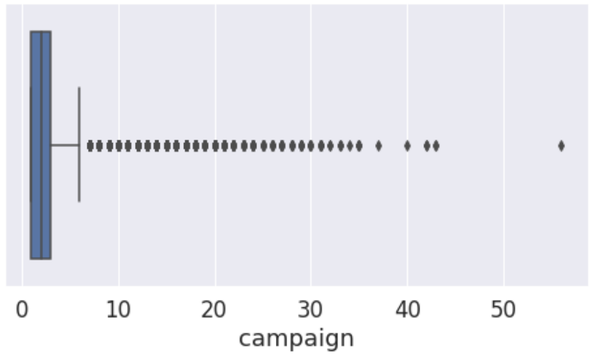
Duration

```
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['duration'])
plt.show()
```



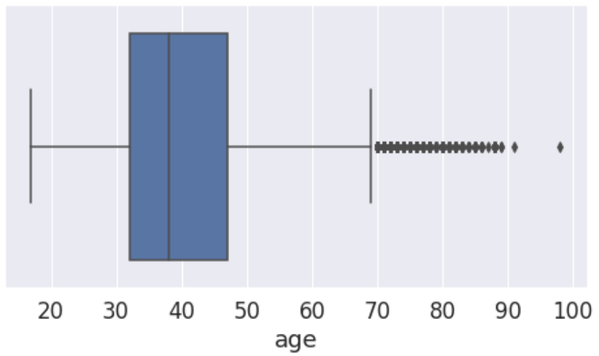
Campaign

```
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['campaign'])
plt.show()
```



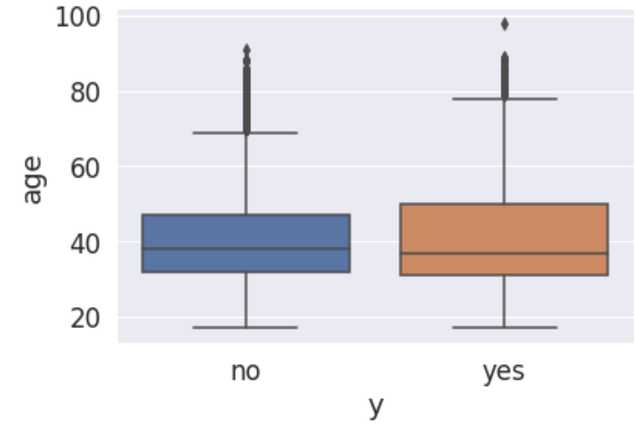
Age

```
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['age'])
plt.show()
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f696bd087d0>
```



Although it can be observed that clients over the age of 45 are more likely to subscribe to the term deposit, there is hardly any difference

Appendix 2

EDA

Calculation of Inter Quartile Range and removal of outliers

```
[32] Q1_d = df['duration'].quantile(.25)
     Q3_d = df['duration'].quantile(.75)
     Q1_a = df['age'].quantile(.25)
     Q3_a = df['age'].quantile(.75)
     Q1_c = df['campaign'].quantile(.25)
     Q3_c = df['campaign'].quantile(.75)
```

```
[33] IQR_d = Q3_d - Q1_d
     IQR_a = Q3_a - Q1_a
     IQR_c = Q3_c - Q1_c
```

```
[34] print(IQR_d)
     print(IQR_a)
     print(IQR_c)
```

```
216.0
15.0
2.0
```

```
[35] lower_d = Q1_d - 1.5 * IQR_d
     upper_d = Q3_d + 1.5 * IQR_d
     lower_a = Q1_a - 1.5 * IQR_a
     upper_a = Q3_a + 1.5 * IQR_a
     lower_c = Q1_c - 1.5 * IQR_d
     upper_c = Q3_c + 1.5 * IQR_d
```

```
print(lower_d,upper_d)
print(lower_a,upper_a)
print(lower_c,upper_c)
```

```
-221.0 643.0
9.5 69.5
-323.0 327.0
```

```
[37] df_out = df[df['duration'] >= lower_d]
     df_out= df[df['duration'] <= upper_d]
     df_out = df[df['age'] >= lower_a]
     df_out= df[df['age'] <= upper_a]
     df_out = df[df['campaign'] >= lower_c]
     df_out= df[df['campaign'] <= upper_c]
```

```
[38] df_out.describe()
```

```
df_out.describe()
```

	age	duration	campaign	pdays	previous
count	32877.000000	32877.000000	32877.000000	32877.000000	32877.000000
mean	40.012167	258.022812	2.559571	962.091097	0.174499
std	10.401993	258.903818	2.749534	187.856729	0.498870
min	17.000000	0.000000	1.000000	0.000000	0.000000
25%	32.000000	103.000000	1.000000	999.000000	0.000000
50%	38.000000	180.000000	2.000000	999.000000	0.000000
75%	47.000000	319.000000	3.000000	999.000000	0.000000
max	98.000000	4918.000000	56.000000	999.000000	7.000000

We assume that

'duration' above 643.0 are outliers

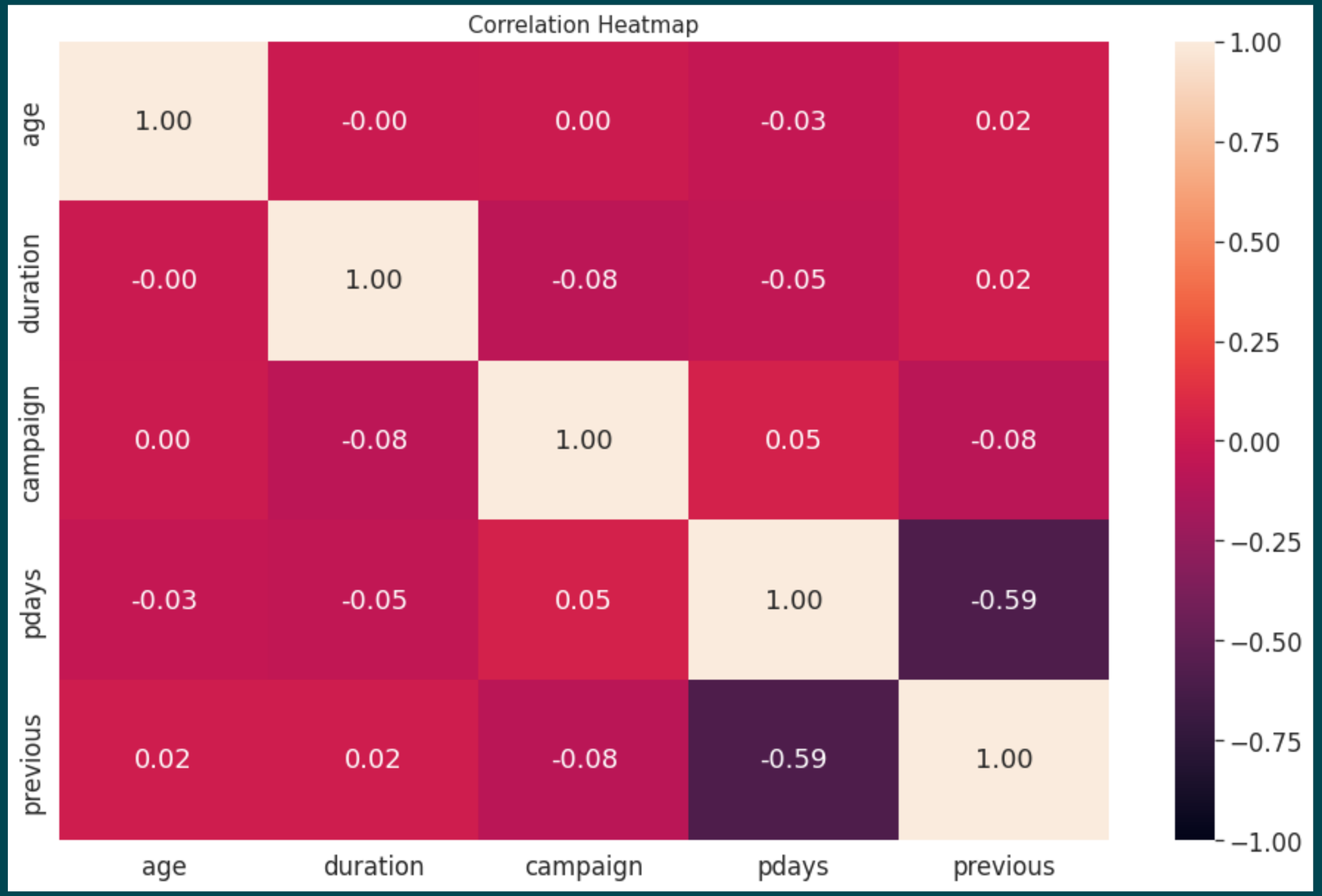
'age' above 69.5 is an outlier

'campaign' above 327.0 is an outlier

Now we drop these outliers

Appendix 2

EDA



2. Checking for correlation between numerical features

```
[39] corr_matrix = df.corr()
     print(corr_matrix)
```

	age	duration	campaign	pdays	previous
age	1.000000	-0.002484	0.002786	-0.032682	0.021126
duration	-0.002484	1.000000	-0.075795	-0.046965	0.022112
campaign	0.002786	-0.075795	1.000000	0.053674	-0.079039
pdays	-0.032682	-0.046965	0.053674	1.000000	-0.589771
previous	0.021126	0.022112	-0.079039	-0.589771	1.000000

Previous and Pdays seems to be correlated, but overall no strong correlation was observed between numerical variables

Appendix 2

EDA

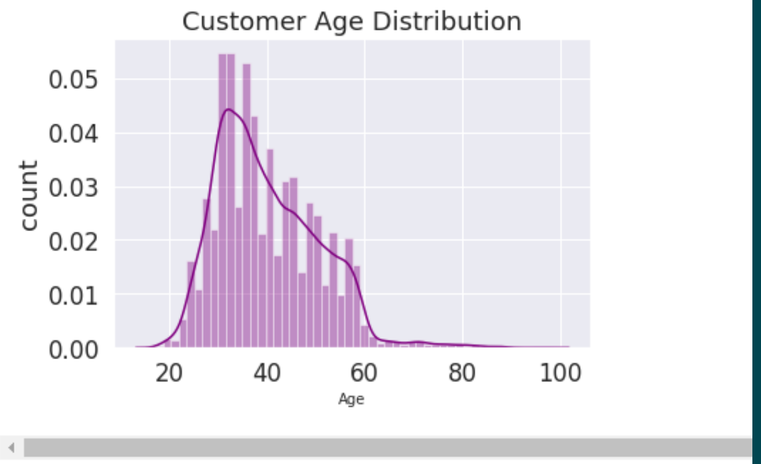
3. Changing 'yes' to 1 and 'no' to 0 in output variable y

```
df['y'] = (df['y']=='yes').astype(int)
df.y.value_counts()
```

4. Which age group has subscribed the most to the term deposit?

```
[42] sns.distplot(df['age'], color = 'purple')
     plt.title('Customer Age Distribution', fontsize = 18)
     plt.xlabel('Age', fontsize = 10)
     plt.ylabel('count')
     plt.show()
```

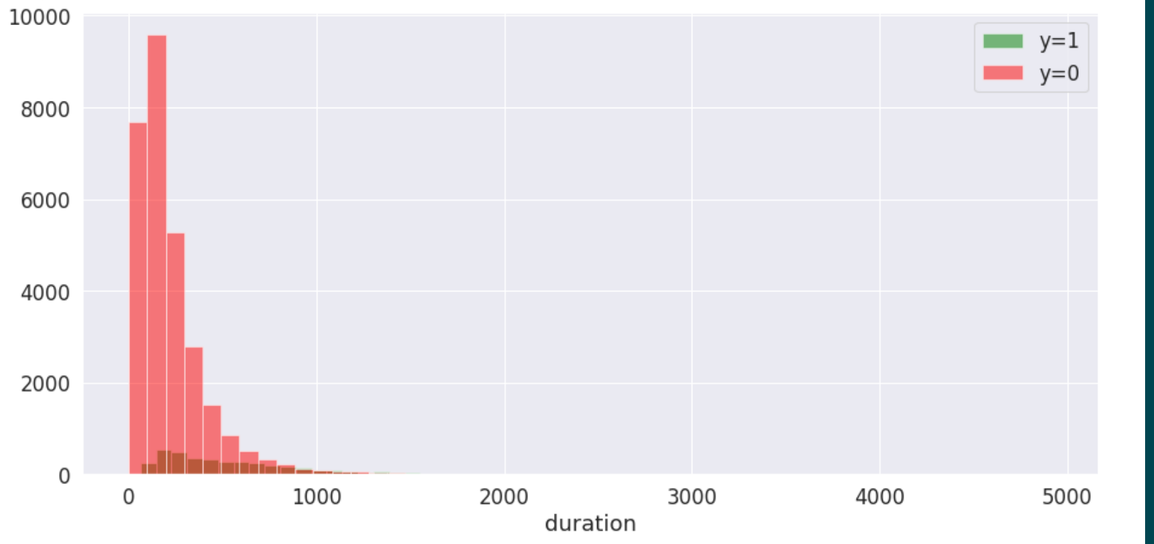
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619
  warnings.warn(msg, FutureWarning)
```



Clients between the age of 25 to 39 hyave subscribed to the deposit the most

```
[43] plt.figure(figsize=(15,7))
     df[df['y']==1]['duration'].hist(alpha = 0.5, color = 'green', bins= 50, label='y=1')
     df[df['y']==0]['duration'].hist(alpha = 0.5, color = 'red', bins= 50, label='y=0')
     plt.legend()
     plt.xlabel('duration')
```

```
Text(0.5, 0, 'duration')
```

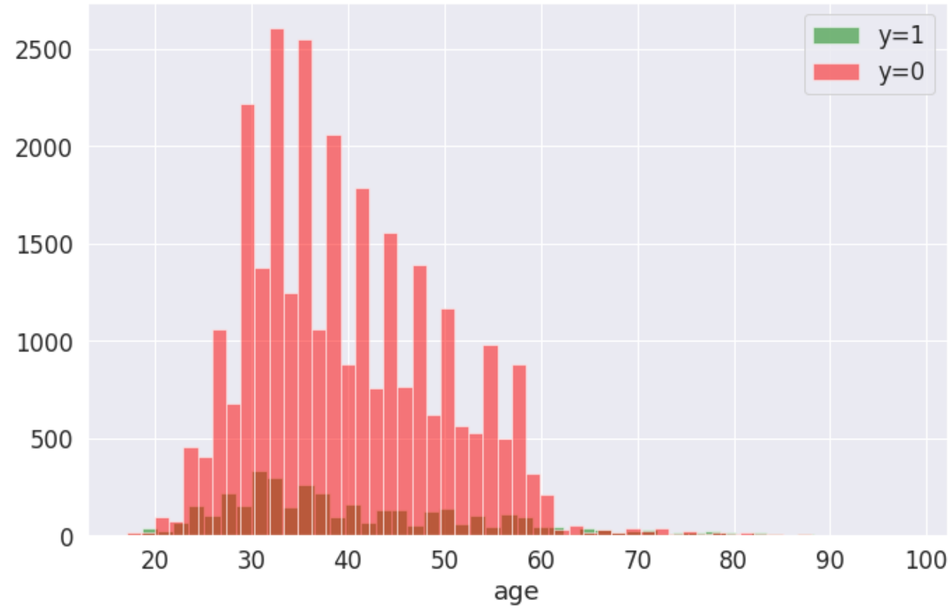


Appendix 2

EDA

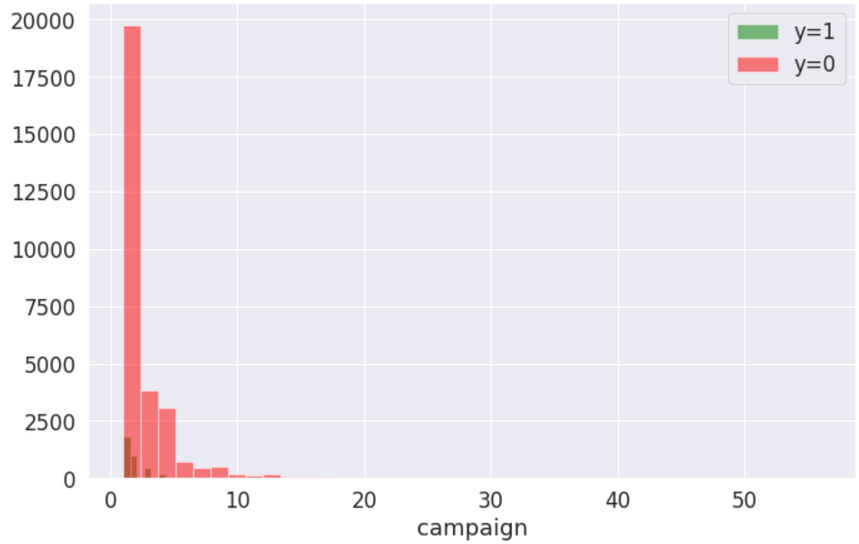
```
plt.figure(figsize=(11,7))
df[df['y']==1]['age'].hist(alpha = 0.5, color = 'green', bins= 50, label='y=1')
df[df['y']==0]['age'].hist(alpha = 0.5, color = 'red', bins= 50, label='y=0')
plt.legend()
plt.xlabel('age')
```

```
Text(0.5, 0, 'age')
```




```
plt.figure(figsize=(11,7))
df[df['y']==1]['campaign'].hist(alpha = 0.5, color = 'green', bins= 40, label='y=1')
df[df['y']==0]['campaign'].hist(alpha =0.5, color = 'red', bins= 40, label='y=0')
plt.legend()
plt.xlabel('campaign')
```

```
Text(0.5, 0, 'campaign')
```



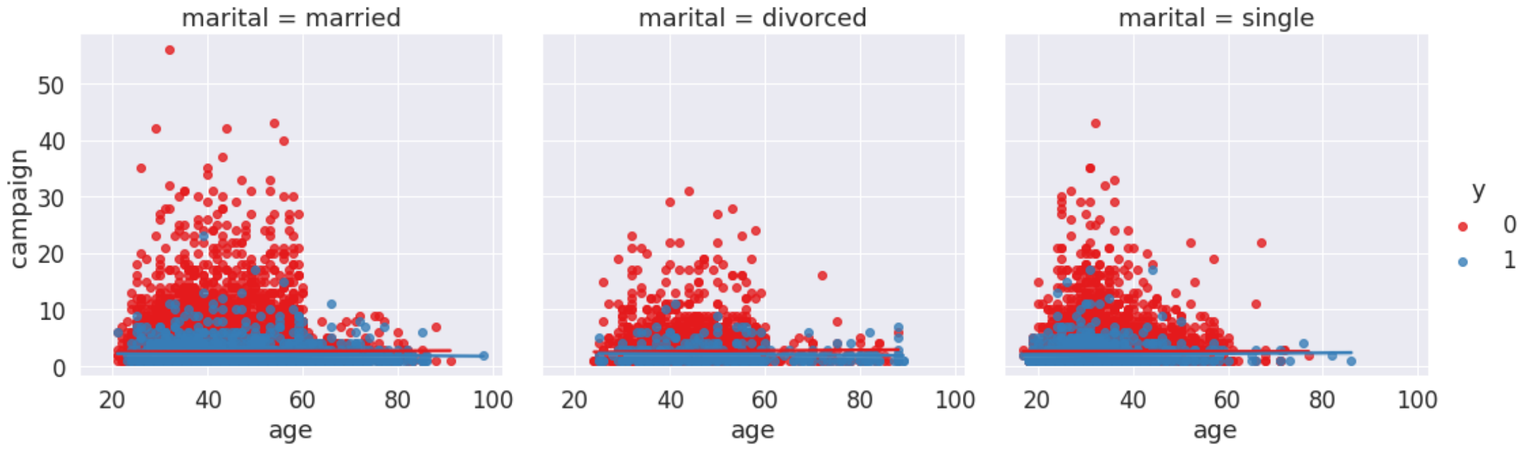
Appendix 2

EDA

5. Checking the trend between marital and y

```
[46] plt.figure(figsize=(11,7))
     sns.lmplot(y='campaign',x='age',hue = 'y', data=df,col='marital',palette='Set1')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f696c5655d0>
<Figure size 792x504 with 0 Axes>
```

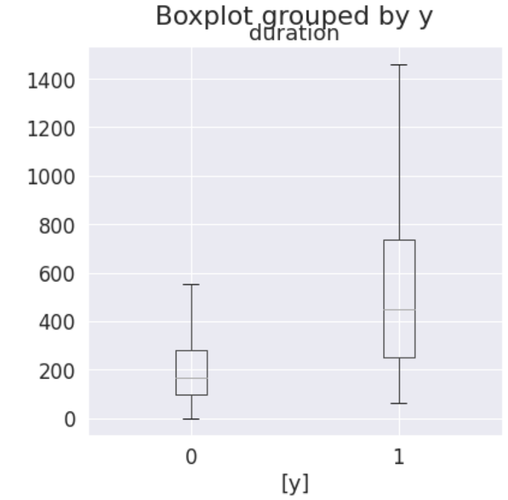


It can be observed that married clients have responded more to the campaigns and subscribed more as well

6. Relationship between the output variable(y) and duration

```
df[['duration', 'y']].boxplot(by=['y'], sym ='', figsize = [6, 6])
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
<matplotlib.axes._subplots.AxesSubplot at 0x7f696bd5a5d0>
```



Appendix 2

EDA

7. Analysis of categorical varibles

```
[48] categorical_features=[feature for feature in df.columns if ((df[feature].dtypes=='O') & (feature not in ['y']))]
     categorical_features
```

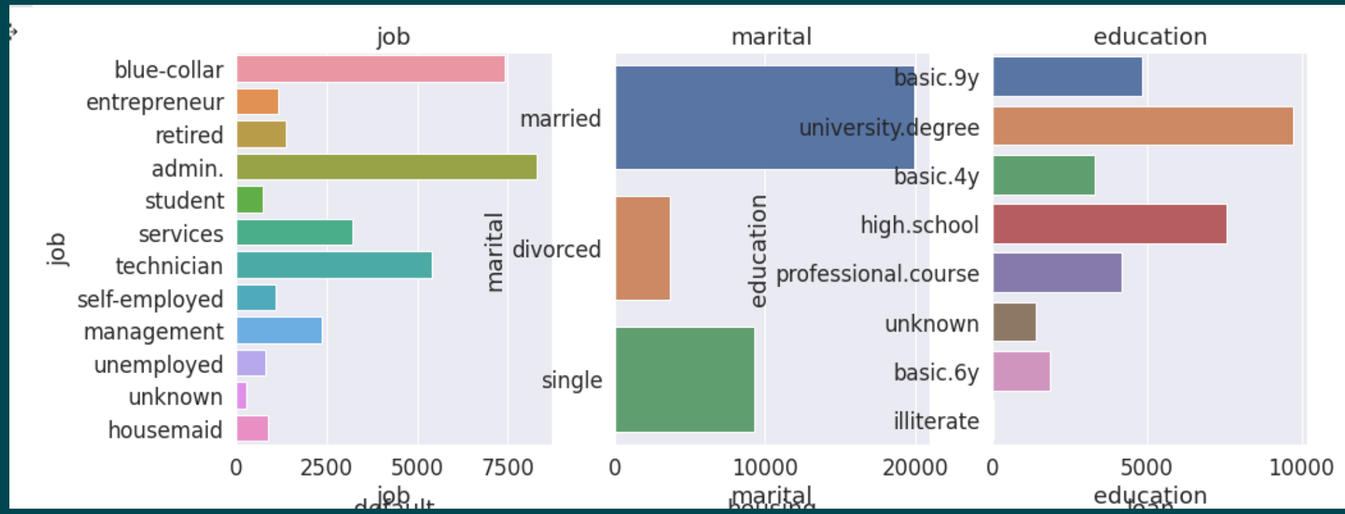
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome']
```

```
for feature in categorical_features:
    print('The feature is {} and number of categories are {}'.format(feature,len(df[feature].unique())))
```

```
The feature is job and number of categories are 12
The feature is marital and number of categories are 3
The feature is education and number of categories are 8
The feature is default and number of categories are 3
The feature is housing and number of categories are 3
The feature is loan and number of categories are 3
The feature is contact and number of categories are 2
The feature is month and number of categories are 10
The feature is day_of_week and number of categories are 5
The feature is poutcome and number of categories are 3
```

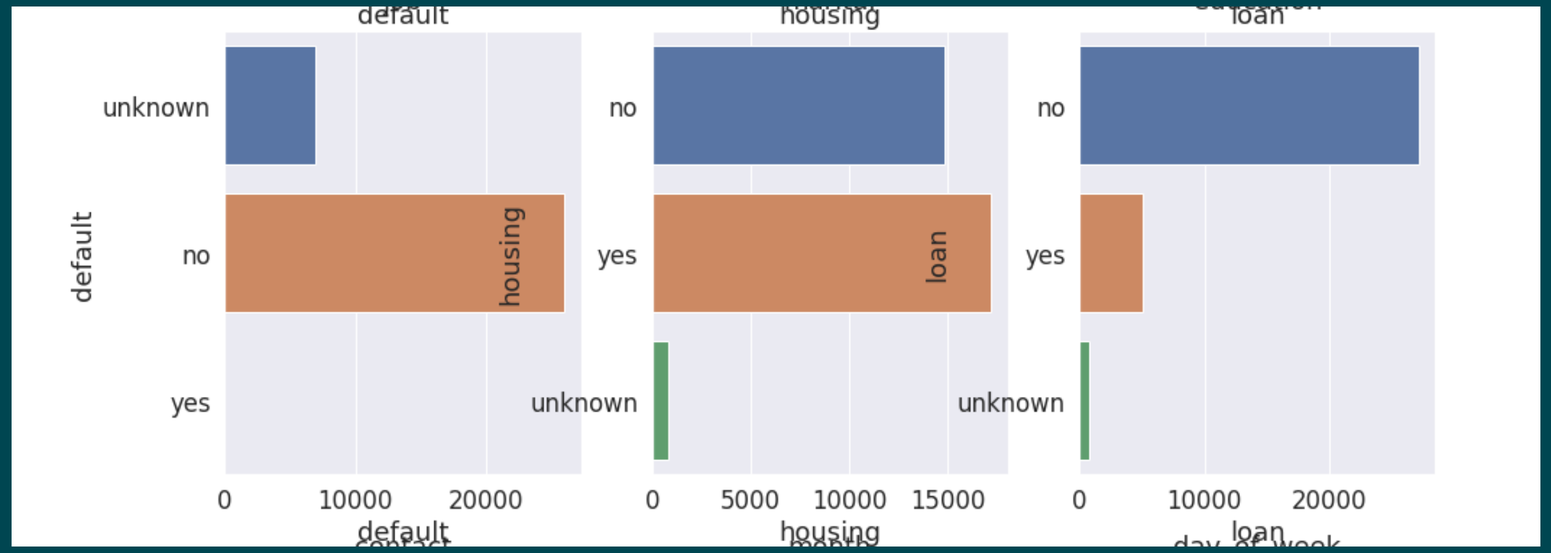
```
[50] #check count based on categorical features
     plt.figure(figsize=(15,80), facecolor='white')
     plotnumber =1
     for categorical_feature in categorical_features:
         ax = plt.subplot(12,3,plotnumber)
         sns.countplot(y=categorical_feature,data=df)
         plt.xlabel(categorical_feature)
         plt.title(categorical_feature)
         plotnumber+=1
```

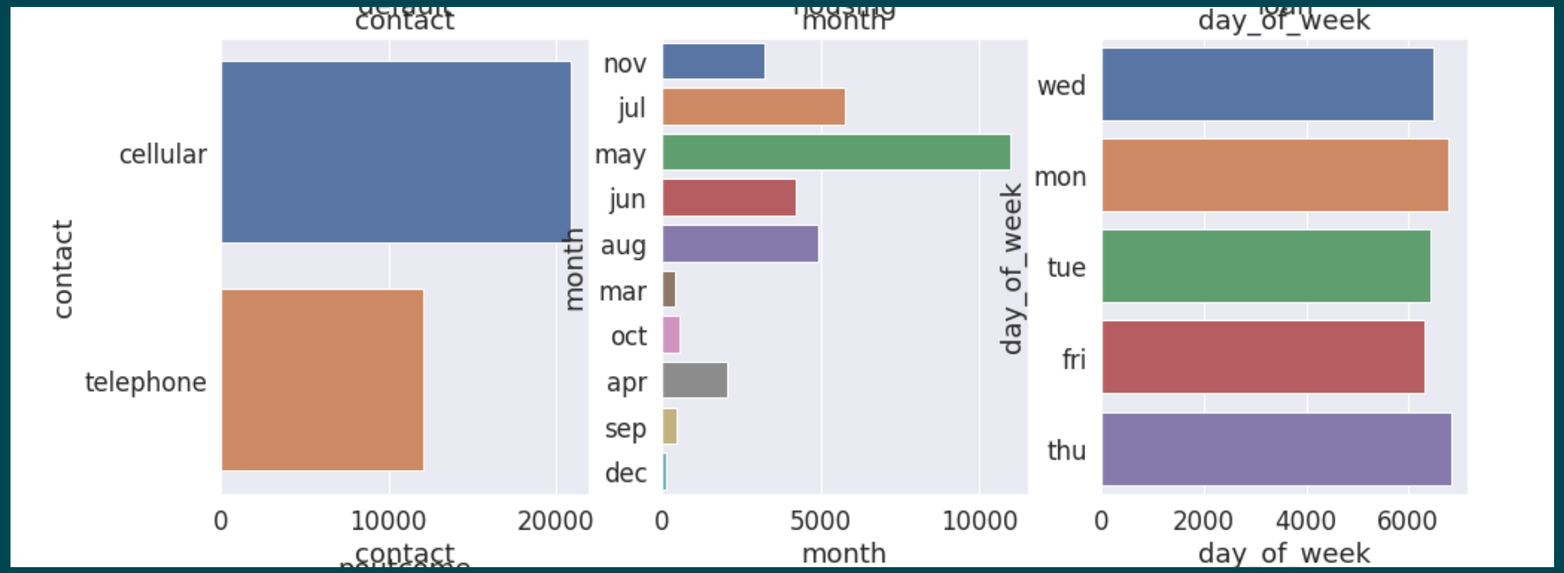
```
plt.show()
```



Appendix 2

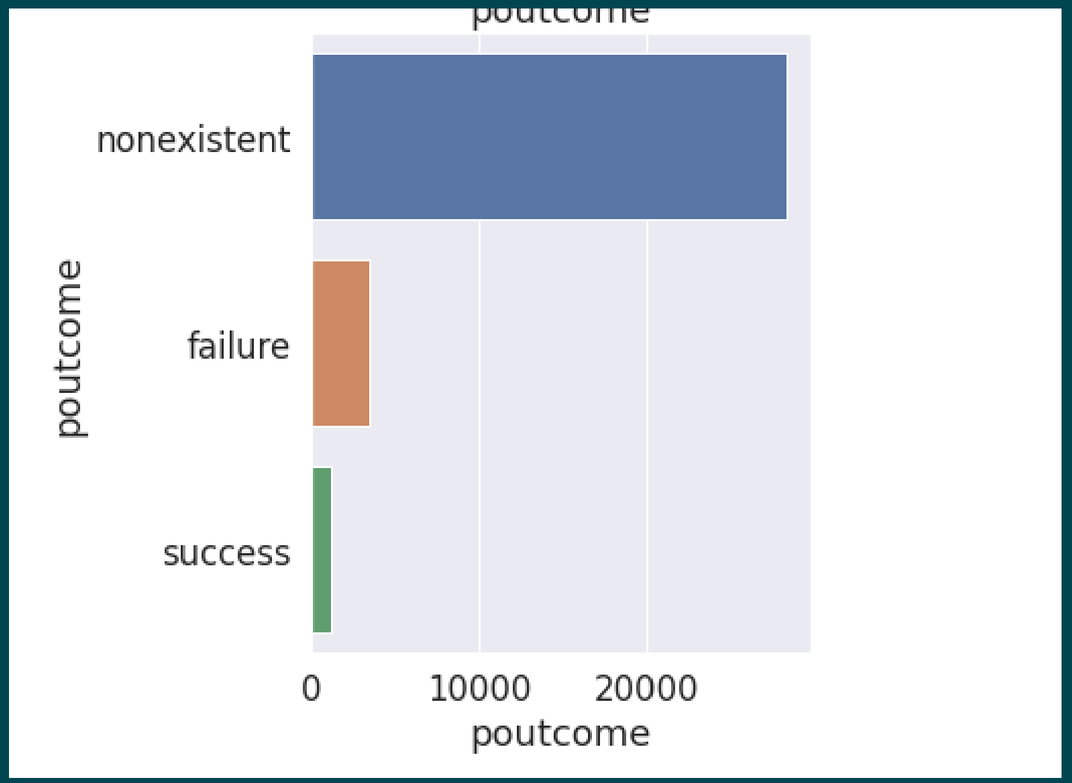
EDA

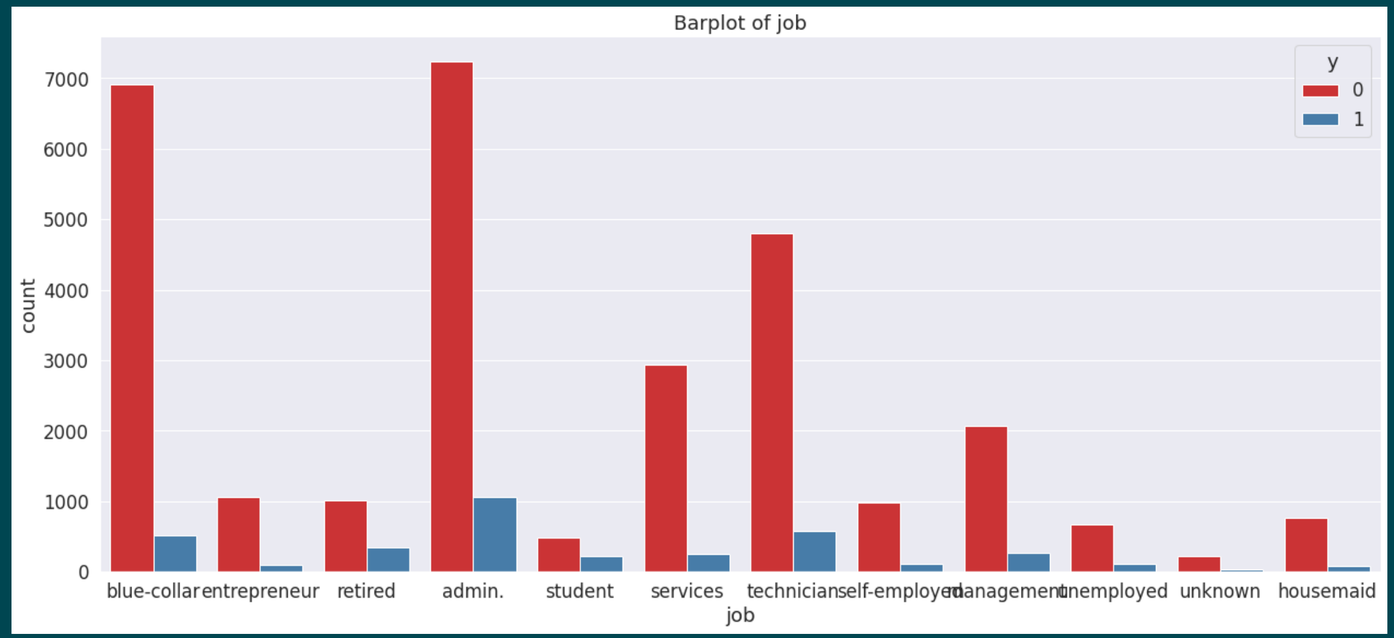




Appendix 2

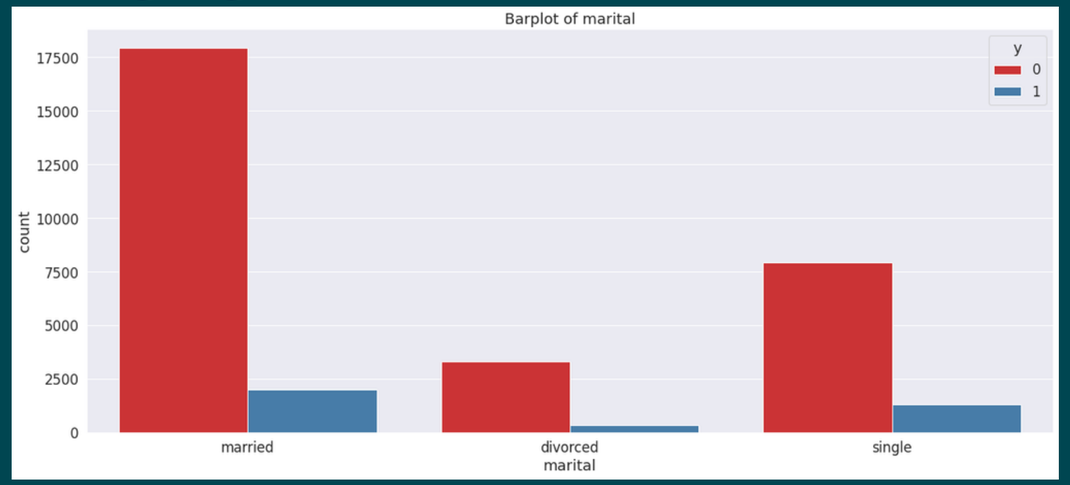
EDA

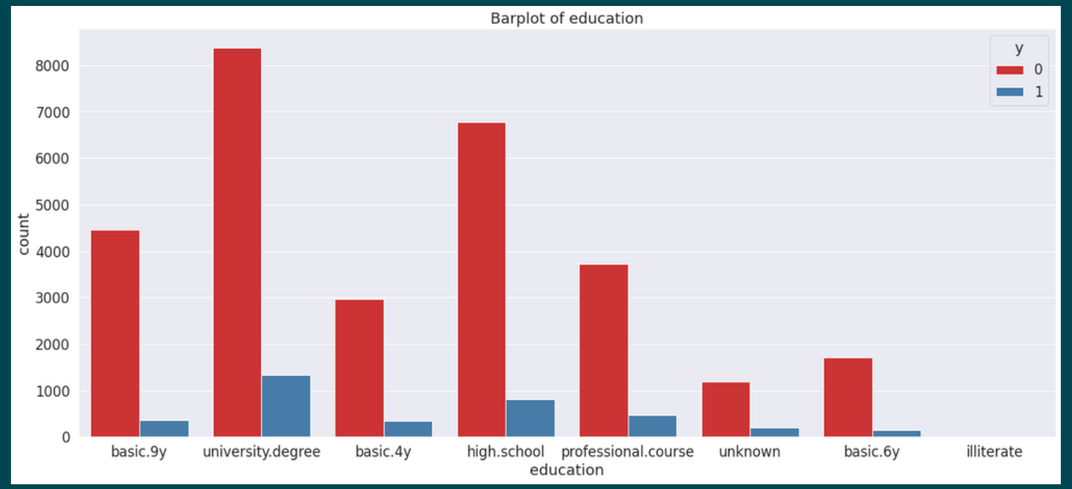


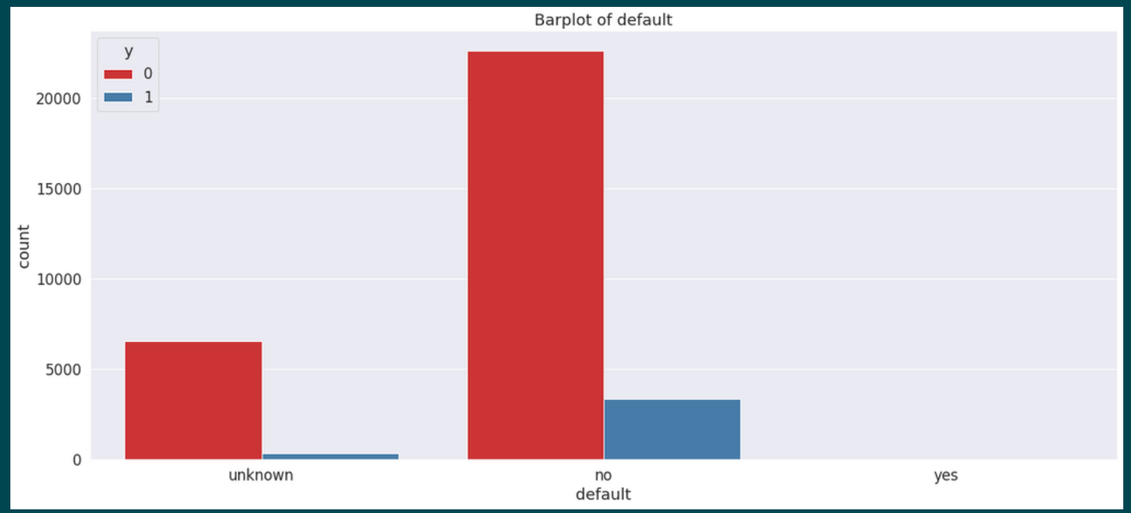


Appendix 2

EDA

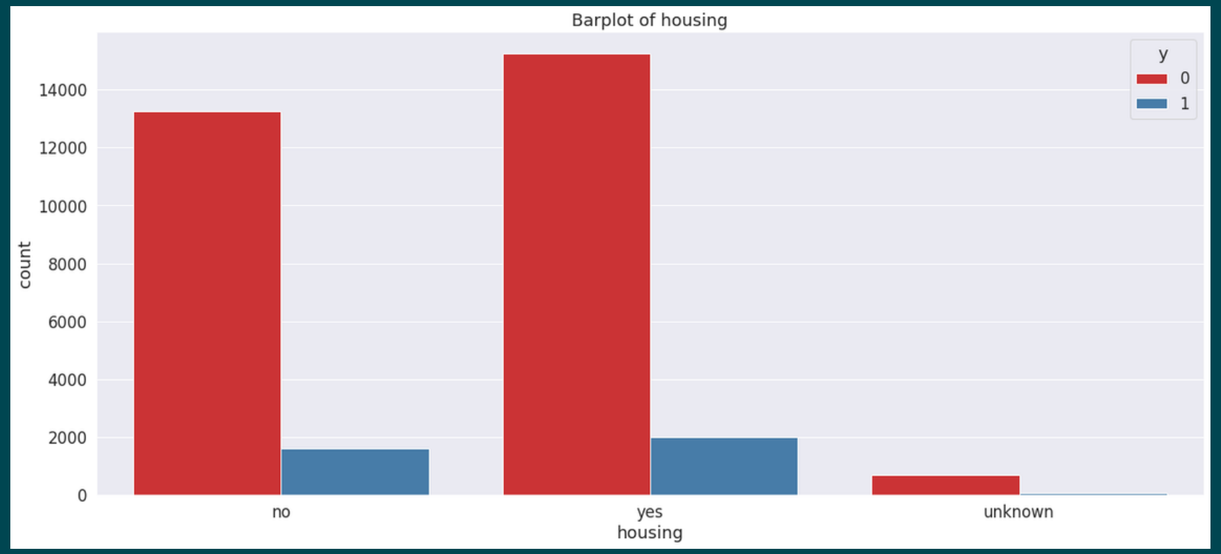


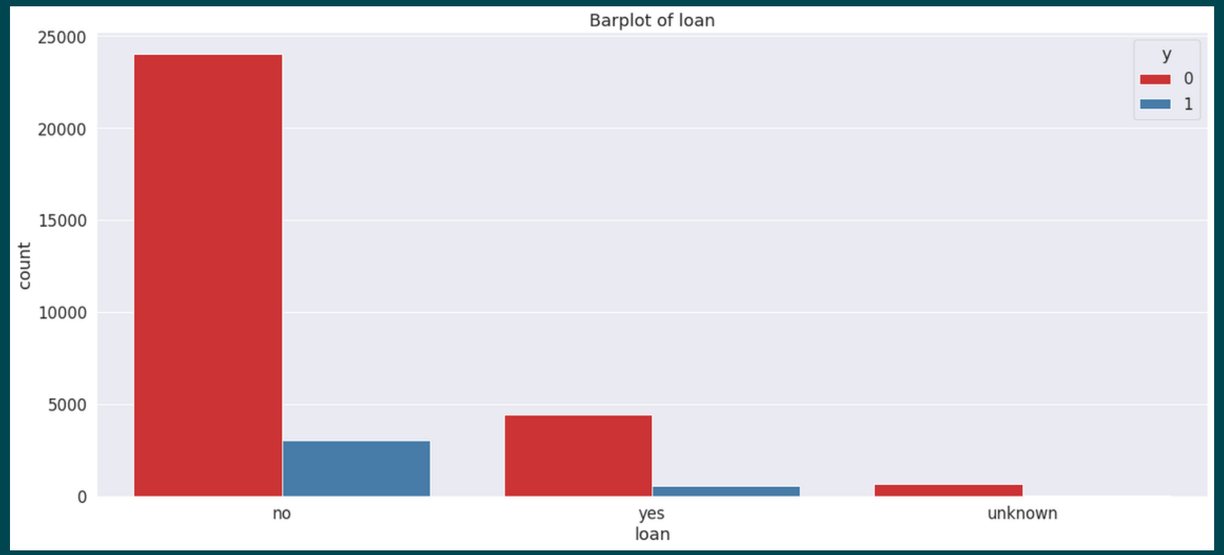


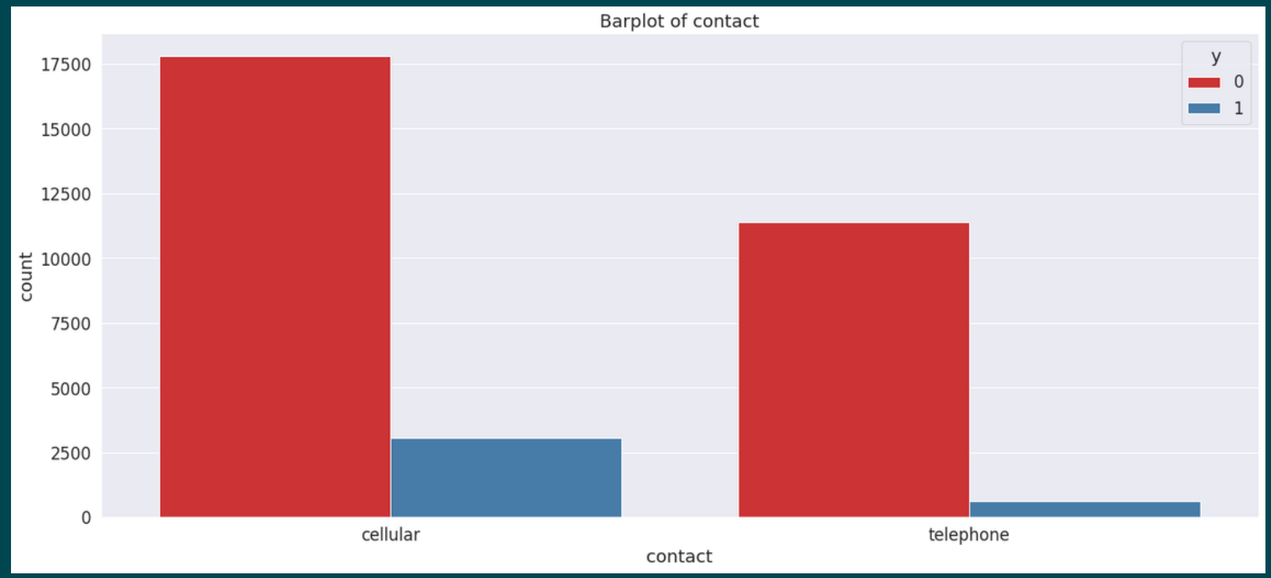


Appendix 2

EDA

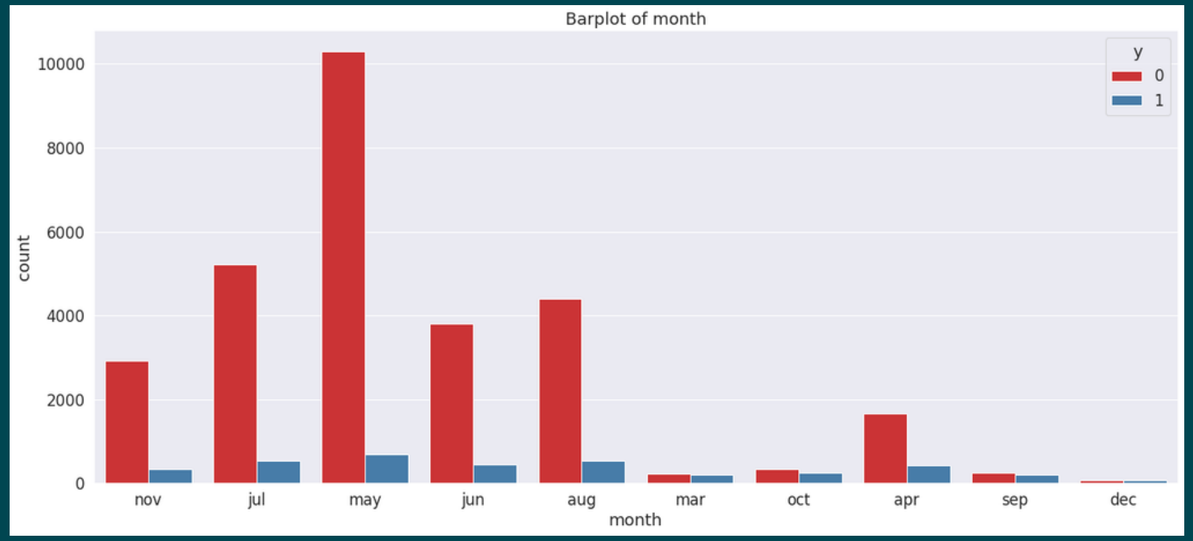


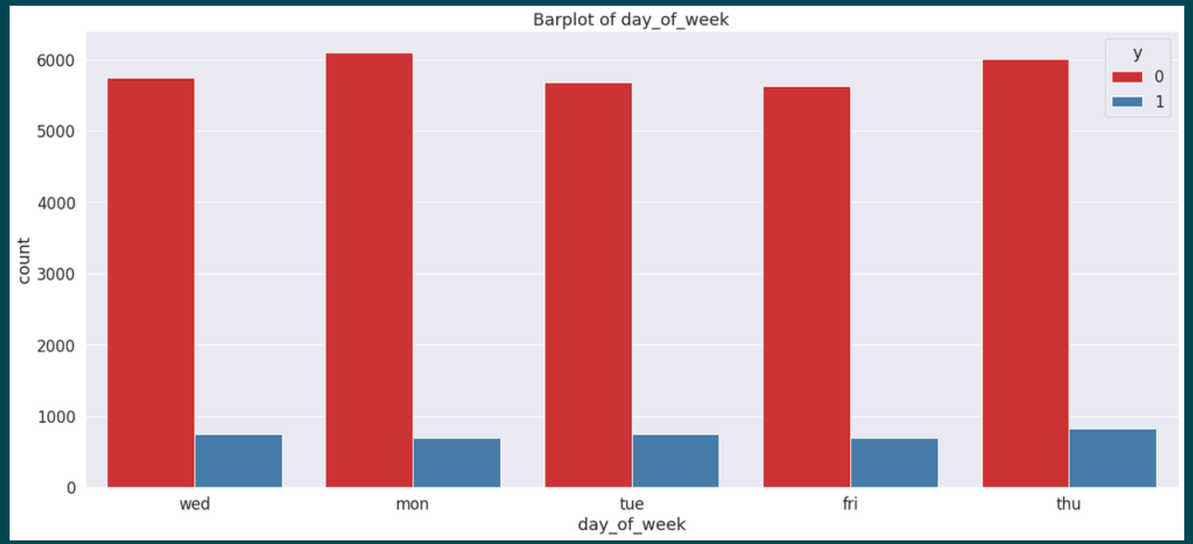


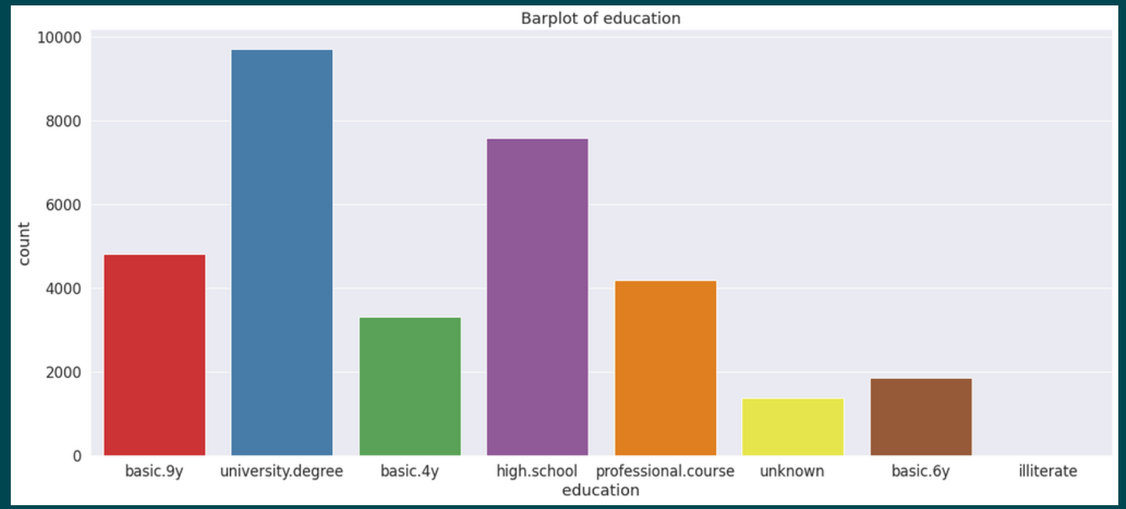


Appendix 2

EDA

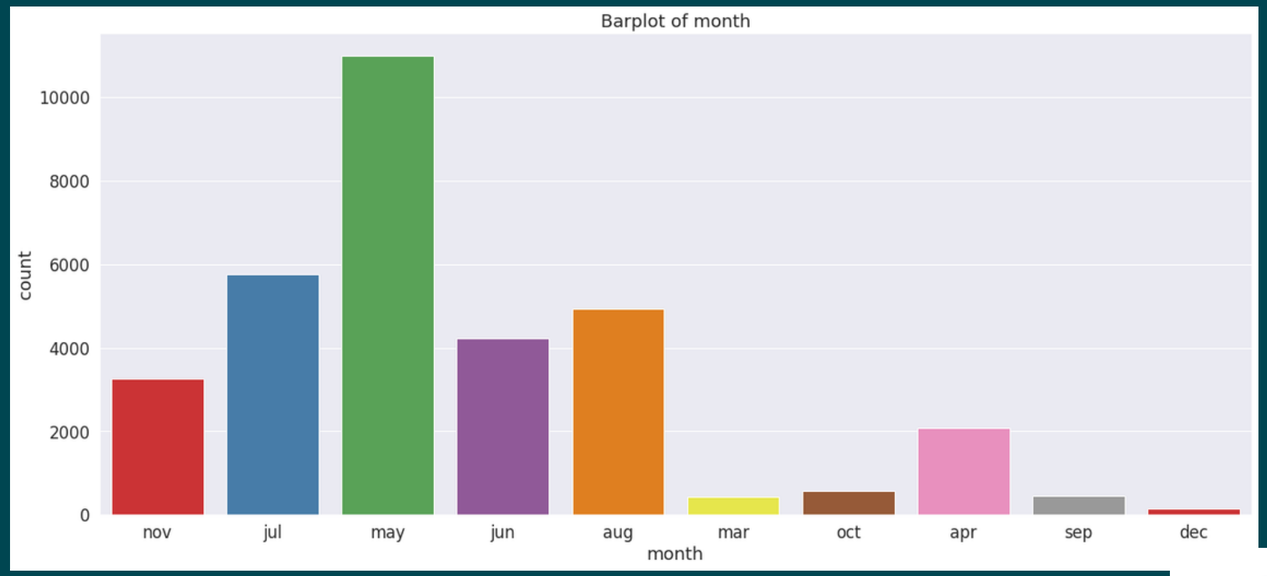


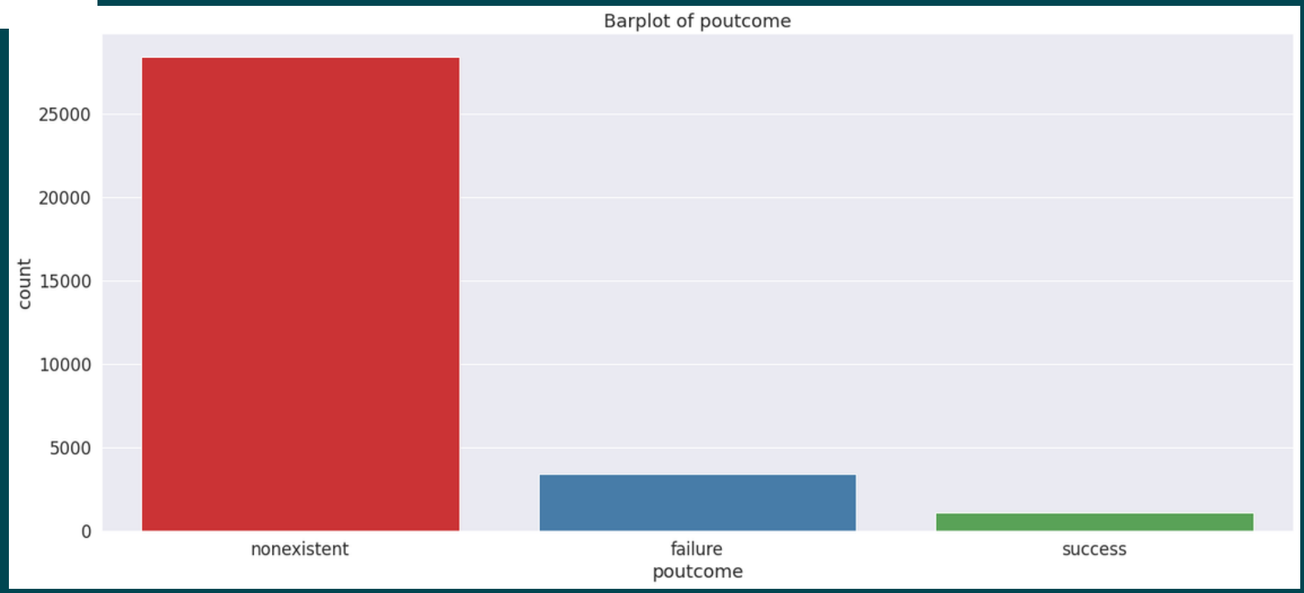




Appendix 2

EDA





Appendix 2

EDA

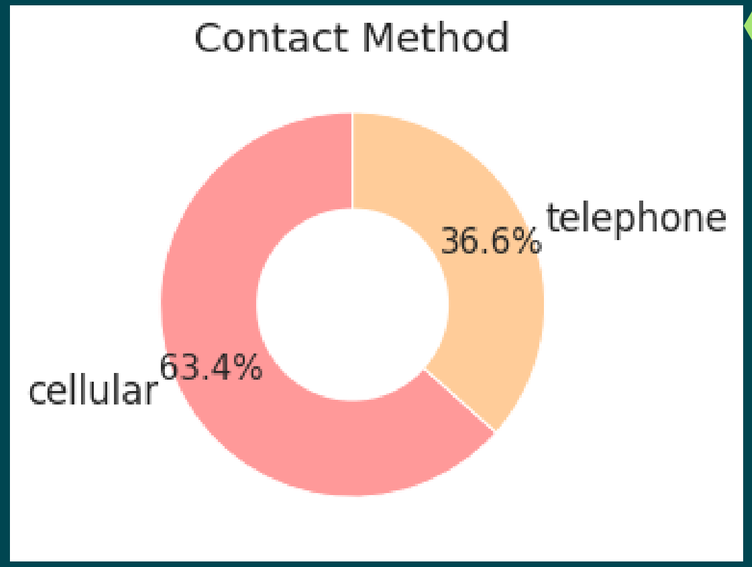
```
#Checking data distribution for categorical value using pie chart
# Data to plot
labels_house = ['yes', 'no', 'unknown']
sizes_house = [17221,14861, 795]
colors_house = ['#ff6666', '#ffcc99', '#ffb3e6']

labels_loan = ['yes', 'no', 'unknown']
sizes_loan = [5012, 27070, 795]
colors_loan = ['#c2c2f0','#ffb3e6', '#66b3ff' ]

labels_contact = ['cellular', 'telephone']
sizes_contact = [20856, 12021]
colors_contact = ['#ff9999','#ffcc99']

labels_default = ['no','unknown','yes']
sizes_default = [25945, 6929, 3]
colors_default = ['#99ff99','#66b3ff','#ff6666' ]


# Plot
plt.rcParams.update({'font.size': 15})

plt.figure(0)
plt.pie(sizes_house, labels=labels_house, colors=colors_house, autopct='%1.1f%%', startangle=90, pctdistance=0.8)
plt.title ('Housing Loan')
centre_circle = plt.Circle((0,0),0.5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()

plt.figure(1)
plt.pie(sizes_loan,labels=labels_loan, colors=colors_loan, autopct='%1.1f%%',startangle=90,pctdistance=0.8)
plt.title ('Personal Loan')
centre_circle = plt.Circle((0,0),0.5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```

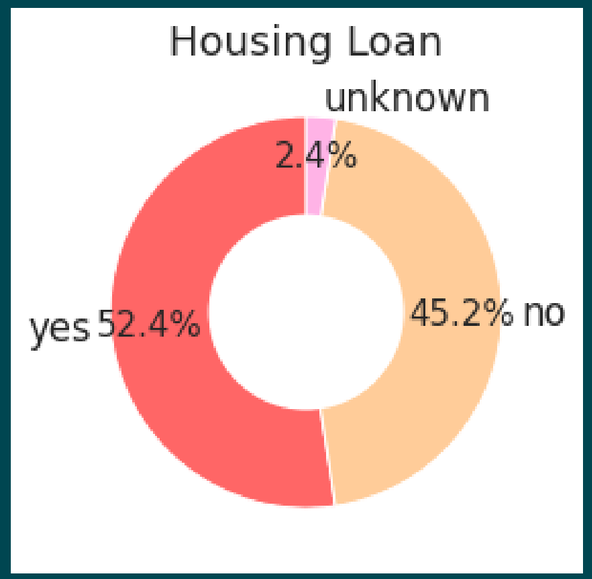
```
plt.figure(2)
plt.pie(sizes_contact, labels=labels_contact, colors=colors_contact, autopct='%1.1f%%', startangle=90,pctdistance=0.8)
plt.title ('Contact Method')
centre_circle = plt.Circle((0,0),0.5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()

plt.figure(3)
plt.pie(sizes_default, labels=labels_default, colors=colors_default, autopct='%1.1f%%', startangle=90,pctdistance=0.8)
plt.title ('default')
centre_circle = plt.Circle((0,0),0.5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```

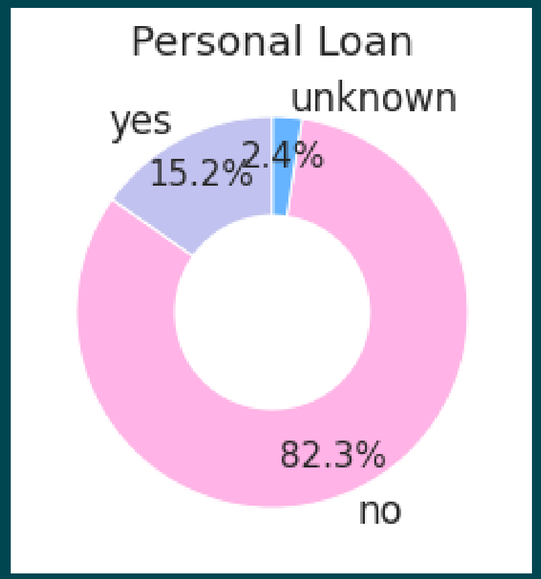
Appendix 2

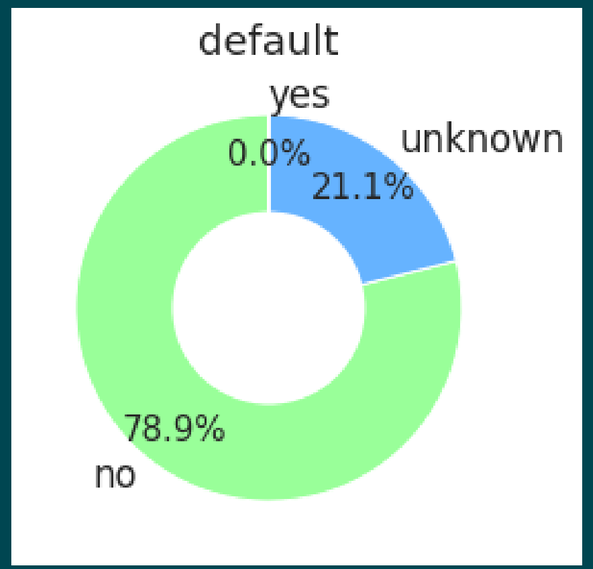
EDA

Most of the clients have a housing loan and most of the clients were contacted through cellular means









Appendix 2

EDA

Inferences

Most of the people in the dataset work as an “admin”

client who married are high in records in given dataset and divorced are less

client who have graduated university are in high numbers in given dataset

data in month of May is high and less in December

Most of the marketing campaign have mostly not made any difference and the success rate is very low

Although communication through cellular mode is higher, there has been decent communication through both modes

Appendix 2

Dummification

Dummification

```
ND = pd.get_dummies(df[['job', 'marital','education','default','housing','loan','contact','month','day_of_week','poutcome']], drop_first=True)
ND
```

	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technician	job_unemployed	job_unknown	marital_married	marital_single	education_basic.6y	education_basic.9y
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32945	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
32946	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
32947	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
32948	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
32949	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

32877 rows × 42 columns

```
[64] a1 = df['y']
     a2 = df['age']
     a3 = df['duration']
     a4 = df['campaign']
     CD = pd.concat([a1,a2,a3,a4, ND], axis=1)
```

```
[65] CD['y']  = CD['y'].replace('yes',1)
     CD['y']  = CD['y'].replace('no',0)
```

Appendix 3

MODEL 1

Logistic Regression

```
[150] X = CD.drop(['y','marital_married', 'marital_single',
                   'loan_unknown', 'loan_yes','campaign'], axis='columns')
      Y = CD['y']
      test_size = 0.30
      seed = 15
      X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
      type(X_train)
```

```
pandas.core.frame.DataFrame
```

```
[151] # Fit the model on 30%
      model = LogisticRegression()
      model.fit(X_train, y_train)
      y_predict = model.predict(X_test)
      LR1 = model.score(X_test, y_test)
      print('Accuracy:',LR1)
      print('confusion_matrix:')
      print(metrics.confusion_matrix(y_test, y_predict))
      A=LR1  # Accuracy of Logistic regression model
```

```
Accuracy: 0.9096715328467153
confusion_matrix:
[[8582  213]
 [ 678  391]]
```

Accuracy Score

```
[152] accuracy_score(y_test,y_predict)
```

```
0.9096715328467153
```

Appendix 3

MODEL 2

Logistic Regression

```
[153] X = CD.drop(['y','marital_married', 'marital_single',
            'loan_unknown', 'loan_yes','campaign','housing_unknown', 'housing_yes','day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
            'day_of_week_wed','age'], axis='columns')
      Y = CD['y']
      test_size = 0.30 # taking 70:30 training and test set
      seed = 15  # Random numbmer seeding for reapeatability of the code
      X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed) # To set the random state
      type(X_train)
```

```
pandas.core.frame.DataFrame
```

```
# Fit the model on 30%
model = LogisticRegression()
model.fit(X_train, y_train)
y_predict = model.predict(X_test)
LR2 = model.score(X_test, y_test)
print('Accuracy:',LR2)
print('confusion_matrix:')
print(metrics.confusion_matrix(y_test, y_predict))
A=LR2  # Accuracy of Logistic regression model
```

```
Accuracy: 0.9090632603406326
confusion_matrix:
[[8578  217]
 [ 680  389]]
```

Accuracy Score

```
accuracy_score(y_test,y_predict)
```

```
0.9090632603406326
```

Appendix 4

Code used for all models

ANN

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(10, activation='relu'),
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.SGD(lr=0.01),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model

history= model.fit(x_train, y_train, epochs= 200)
```

```
822/822 [==============================] - 2s 3ms/step - loss: 0.2713 - accuracy: 0.8906 - precision: 0.5383 - a=recall: 0.2088
Epoch 164/200
822/822 [==============================] - 2s 2ms/step - loss: 0.2713 - accuracy: 0.8910 - precision: 0.5456 - a=recall: 0.1976
Epoch 165/200
822/822 [==============================] - 2s 2ms/step - loss: 0.2727 - accuracy: 0.8910 - precision: 0.5437 - a=recall: 0.2057
Epoch 166/200
```

```
72] model.summary()

Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 10)                460

 dense_31 (Dense)            (None, 7)                 77

 dense_32 (Dense)            (None, 5)                 40

 dense_33 (Dense)            (None, 1)                 6

=================================================================
Total params: 583
Trainable params: 583
Non-trainable params: 0
_________________________________________________________________
```

Appendix 4

ANN

```
predictions=model.predict(x_test)
predictions=np.rint(predictions)
predictions
```

```
array([[0.],
       [0.],
       [0.],
       ...,
       [0.],
       [0.],
       [0.]], dtype=float32)
```

```
[74] print(classification_report(y_test,predictions))
     print(metrics.confusion_matrix(y_test,predictions))
```

```
                precision    recall  f1-score   support

             0       0.91      0.98      0.94      5840
             1       0.62      0.21      0.31       736

      accuracy                           0.90      6576
     macro avg       0.77      0.60      0.63      6576
  weighted avg       0.88      0.90      0.87      6576

  [[5748   92]
   [ 583  153]]
```

Confusion Matrix

Adam

Optimizer	Architecture	Epochs	Accuracy	Precision	Recall	Loss
Adam	10-7-5-1	50	0.9046	0.5943	0.4857	0.2095
Adam	10-7-5-1	100	0.9059	0.6015	0.4887	0.2086
Adam	10-7-5-1	107	0.9078	0.6091	0.5076	0.2075
Adam	10-7-5-1	150	0.9078	0.6101	0.5035	0.2067
Adam	10-7-5-1	200	0.9057	0.5942	0.5147	0.2066
Adam	45-2-1	50	0.904	0.5821	0.5261	0.2129
Adam	45-2-1	53	0.906	0.5965	0.513	0.2118
Adam	45-2-1	57	0.9058	0.5958	0.5106	0.2095
Adam	45-2-1	100	0.9051	0.5861	0.5383	0.2069
Adam	45-2-1	114	0.9072	0.6012	0.5258	0.2046
Adam	45-2-1	150	0.9072	0.5968	0.5457	0.2034
Adam	45-2-1	164	0.9074	0.5984	0.5423	0.2036
Adam	45-2-1	165	0.9074	0.5969	0.5497	0.2038
Adam	45-2-1	200	0.9058	0.5861	0.5589	0.2026
Adam	40-5-1	56	0.9051	0.5913	0.5123	0.2124
Adam	40-5-1	76	0.9063	0.5929	0.5379	0.2076
Adam	40-5-1	114	0.9069	0.597	0.5366	0.2065
Adam	40-5-1	179	0.9073	0.5974	0.544	0.2026
Adam	40-5-1	200	0.9059	0.5887	0.5484	0.204
Adam	35-5-5-1	55	0.906	0.5963	0.5137	0.2125
Adam	35-5-5-1	82	0.9061	0.5946	0.5245	0.2095
Adam	35-5-5-1	155	0.906	0.5865	0.5649	0.2059
Adam	35-5-5-1	196	0.9065	0.5903	0.5568	0.2051
Adam	35-5-5-1	200	0.9023	0.567	0.5636	0.2063
Adam	11-10-4-4-3-7-7-1	50	0.8873	0.00E+00	0.00E+00	0.3524
Adam	11-10-4-4-3-7-7-1	100	0.8873	0.00E+00	0.00E+00	0.3524
Adam	11-10-4-4-3-7-7-1	150	0.8873	0.00E+00	0.00E+00	0.3523
Adam	11-10-4-4-3-7-7-1	200	0.8873	0.00E+00	0.00E+00	0.3522

Max Accuracy:0.9078

SGD

Optimizer	Architecture	Epochs	Accuracy	Precision	Recall	Loss
SGD	10-7-5-1	50	0.8894	0.5441	0.1143	0.3021
SGD	10-7-5-1	100	0.8867	0.4314	0.0148	0.2825
SGD	10-7-5-1	150	0.8866	0.4045	0.0121	0.2739
SGD	10-7-5-1	200	0.8919	0.5497	0.2256	0.2668
SGD	45-2-1	50	0.89	0.5677	0.1032	0.2841
SGD	45-2-1	100	0.8895	0.5541	0.1019	0.2765
SGD	45-2-1	150	0.8892	0.5501	0.0944	0.2771
SGD	45-2-1	200	0.8896	0.551	0.113	0.2621
SGD	40-5-1	50	0.8873	0.00E+00	0.00E+00	0.3114
SGD	40-5-1	100	0.8897	0.5598	0.0995	0.28
SGD	40-5-1	150	0.8888	0.5392	0.0951	0.278
SGD	40-5-1	200	0.8895	0.5483	0.113	0.2647
SGD	35-5-5-1	50	0.8902	0.5739	0.1022	0.2814
SGD	35-5-5-1	100	0.8902	0.5765	0.0978	0.2736
SGD	35-5-5-1	150	0.8898	0.566	0.0954	0.2685
SGD	35-5-5-1	200	0.8889	0.5483	0.0823	0.2622
SGD	11-10-4-4-3-7-7-5	50	0.904	0.6022	0.4381	0.215
SGD	11-10-4-4-3-7-7-6	57	0.9055	0.5984	0.4911	0.214
SGD	11-10-4-4-3-7-7-7	74	0.906	0.6057	0.4762	0.2127
SGD	11-10-4-4-3-7-7-8	86	0.9065	0.6121	0.4658	0.2122
SGD	11-10-4-4-3-7-7-9	111	0.9064	0.6194	0.4408	0.2111
SGD	11-10-4-4-3-7-7-1	121	0.9068	0.6201	0.4476	0.2103
SGD	11-10-4-4-3-7-7-1	124	0.9076	0.6233	0.456	0.212
SGD	11-10-4-4-3-7-7-1	126	0.9078	0.6289	0.4435	0.2111
SGD	11-10-4-4-3-7-7-1	147	0.908	0.623	0.4654	0.2103
SGD	11-10-4-4-3-7-7-1	156	0.9077	0.6155	0.4826	0.2099
SGD	11-10-4-4-3-7-7-1	171	0.9082	0.6273	0.458	0.2085
SGD	11-10-4-4-3-7-7-1	191	0.9083	0.627	0.4597	0.2082
SGD	11-10-4-4-3-7-7-1	207	0.9086	0.63	0.4583	0.2078
SGD	11-10-4-4-3-7-7-1	247	0.9086	0.633	0.4496	0.2071
SGD	11-10-4-4-3-7-7-1	291	0.9086	0.6277	0.4641	0.2062
SGD	11-10-4-4-3-7-7-2	400	0.907	0.6111	0.4806	0.2058

Max Accuracy:0.9086

Appendix 5

Bagging

```
# splitting data into training and test set for independent attributes
from sklearn.model_selection import train_test_split

X_train, X_test, train_labels, test_labels = train_test_split(X, Y, test_size=.30, random_state=1)
```

```
from sklearn.ensemble import BaggingClassifier

bgcl = BaggingClassifier(base_estimator=dt_model, n_estimators=50)

bgcl = bgcl.fit(train_set, train_labels)
```

Prediction and Confusion Matrix

```
y_predict = bgcl.predict(test_set)

BGC=bgcl.score(test_set , test_labels)
print(BGC)

print(metrics.confusion_matrix(test_labels, y_predict))
```

```
0.8786039453717754
[[8675  112]
 [1088   10]]
```

Accuracy Score

```
[218] accuracy_score(test_labels,y_predict)

      0.8786039453717754
```

Appendix 6

Boosting

AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
abcl = AdaBoostClassifier(base_estimator=dt_model, n_estimators=10)

abcl = abcl.fit(train_set, train_labels)
```

Prediction and Confusion Matrix

```
[224] y_predict = abcl.predict(test_set)

ADE=abcl.score(test_set , test_labels)
print(ADE)

print(metrics.confusion_matrix(test_labels, y_predict))
```

```
0.8113302984319676
[[7903  884]
 [ 981  117]]
```

Accuracy Score

Accuracy Score

```
[225] accuracy_score(test_labels,y_predict)
```

```
0.8113302984319676
```

Appendix 6

Boosting

GradientBoost Classifier

```
[ ]  from sklearn.ensemble import GradientBoostingClassifier
     gbcl = GradientBoostingClassifier(n_estimators = 50)
     gbcl = gbcl.fit(train_set, train_labels)
```

Prediction and Confusion Matrix

```
[ ]  y_predict = gbcl.predict(test_set)
     GBC=gbcl.score(test_set , test_labels)
     print(GBC)
     print(metrics.confusion_matrix(test_labels, y_predict))
```

```
0.8889226100151745
[[8787    0]
 [1098    0]]
```

Accuracy Score

```
0.8889226100151745
```

Appendix 7

Other Models

Decision Tree

```
# Label encoder order in alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
my_data['job']       = labelencoder_X.fit_transform(my_data['job'])
my_data['marital']   = labelencoder_X.fit_transform(my_data['marital'])
my_data['education']= labelencoder_X.fit_transform(my_data['education'])
my_data['default']   = labelencoder_X.fit_transform(my_data['default'])
my_data['housing']   = labelencoder_X.fit_transform(my_data['housing'])
my_data['loan']      = labelencoder_X.fit_transform(my_data['loan'])

my_data['contact']     = labelencoder_X.fit_transform(my_data['contact'])
my_data['month']       = labelencoder_X.fit_transform(my_data['month'])
```

```
[175] #function to creat group of ages, this helps because we have 78 differente values here
      def age(dataframe):
          dataframe.loc[dataframe['age'] <= 32, 'age'] = 1
          dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] <= 47), 'age'] = 2
          dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] <= 70), 'age'] = 3
          dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] <= 98), 'age'] = 4

          return dataframe

      age(my_data);
```

```
[178] 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'] <= 644.5), 'duration'] = 4
          data.loc[data['duration']  > 644.5, 'duration'] = 5

          return data
      duration(my_data);
```

```
my_data.loc[(my_data['pdays'] == 999), 'pdays'] = 1
my_data.loc[(my_data['pdays'] > 0) & (my_data['pdays'] <= 10), 'pdays'] = 2
my_data.loc[(my_data['pdays'] > 10) & (my_data['pdays'] <= 20), 'pdays'] = 3
my_data.loc[(my_data['pdays'] > 20) & (my_data['pdays'] != 999), 'pdays'] = 4
my_data.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
0	3	1	1	2	1	0	0	0	7	wed	3	4	2	0	nonexistent	no
1	2	2	1	6	0	0	0	1	7	wed	3	2	2	1	failure	no
2	4	5	1	0	0	0	0	0	3	mon	5	1	2	0	nonexistent	yes
3	2	0	1	6	0	2	0	1	6	mon	2	2	2	0	nonexistent	no
4	3	5	0	6	0	0	0	0	4	tue	4	2	2	0	nonexistent	no

```
[181] my_data['poutcome'].replace(['unknown', 'failure','other', 'success'], [1,2,3,4], inplace  = True)
```

```
[182] print(my_data.shape)
      my_data.head()
```

```
(32950, 16)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
0	3	1	1	2	1	0	0	0	7	wed	3	4	2	0	nonexistent	no

```
[185] X = Final_data.drop(['y'],axis=1)
      Y = Final_data.y
```

```
[186] for feature in my_data.columns: # Loop through all columns in the dataframe
          if my_data[feature].dtype == 'object': # Only apply for columns with categorical strings
              my_data[feature] = pd.Categorical(my_data[feature]).codes # Replace strings with an integer
```

```
[188] train_char_label = ['No', 'Yes']
```

```
[189] from sklearn.tree import DecisionTreeClassifier
      from sklearn.feature_extraction.text import CountVectorizer

      # splitting data into training and test set for independent attributes
      from sklearn.model_selection import train_test_split

      X_train, X_test, train_labels, test_labels = train_test_split(X, Y, test_size=.30, random_state=1)
```

```
[190] # splitting data into training and test set for independent attributes in the ratio of 70:30
      n=my_data['y'].count()
      train_set = my_data.head(int(round(n*0.7))) # Up to the last initial training set row
      test_set = my_data.tail(int(round(n*0.3))) # Past the last initial training set row

      # capture the target column ("y") into separate vectors for training set and test set
      train_labels = train_set.pop("y")
      test_labels = test_set.pop("y")
```

```
[191] # invoking the decision tree classifier function. Using 'entropy' method of finding the split columns.
      dt_model = DecisionTreeClassifier(criterion = 'entropy' )
```

```
[192] dt_model.fit(train_set, train_labels)

      DecisionTreeClassifier(criterion='entropy')
```

```
[ ]  #Print the accuracy of the model & print the confusion matrix
      dt_model.score(test_set , test_labels)
      test_pred = dt_model.predict(test_set)
```

```
[ ]  print (pd.DataFrame(dt_model.feature_importances_, columns = ["Imp"], index = train_set.columns))#Print the feature importance of the decision model
```

```
y_predict = dt_model.predict(test_set)
```

```
[ ]  print(dt_model.score(train_set , train_labels))
     print(dt_model.score(test_set , test_labels))

     0.9762410578799047
     0.842185128983308
```

```
[ ]  print(metrics.confusion_matrix(test_labels, y_predict))

     [[8022  731]
      [ 829  303]]
```

```
[ ]  accuracy_score(test_labels,y_predict)

     0.842185128983308
```

Dealing with overfitting

```
[193] reg_dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 7)
      reg_dt_model.fit(train_set, train_labels)

      DecisionTreeClassifier(criterion='entropy', max_depth=7)

      print (pd.DataFrame(dt_model.feature_importances_, columns = ["Imp"], index = train_set.columns))
```

```
[195] y_predict = reg_dt_model.predict(test_set)

[196] DTC=reg_dt_model.score(test_set , test_labels)
      print(DTC)

      0.9033889731917046

      print(metrics.confusion_matrix(test_labels, y_predict))

      [[8443  310]
       [ 645  487]]

[198] accuracy_score(test_labels,y_predict)

      0.9033889731917046

[199] print('Precision: %.3f' % precision_score(test_labels,y_predict))
      print('Recall: %.3f' % recall_score(test_labels, y_predict))

      Precision: 0.611
      Recall: 0.430
```

Appendix 7

Other Models

Random Forest

```
[200] from sklearn.ensemble import RandomForestClassifier
      rfcl = RandomForestClassifier(n_estimators = 50)
      rfcl = rfcl.fit(train_set, train_labels)
```

Prediction and Confusion Matrix

```
[201] y_predict = rfcl.predict(test_set)
      RFC=rfcl.score(test_set , test_labels)
      print(RFC)
      print(metrics.confusion_matrix(test_labels, y_predict))
```

```
0.8981284774911482
[[8471  282]
 [ 725  407]]
```

```
[202] accuracy_score(test_labels,y_predict)
```

```
0.8981284774911482
```

Accuracy,Precision and Recall

```
Precision: 0.591
Recall: 0.360
```

Appendix 7

Other Models

KNN: K-Nearest Neighbour

```
[ ] final_data = Final_data[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
                             'contact', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome']]
    final_data.shape
```

```
(32950, 14)
```

```
[ ] from sklearn.preprocessing import StandardScaler
    sc_X = StandardScaler()
    X_train=sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)
```

```
[ ] X_std = pd.DataFrame(StandardScaler().fit_transform(final_data))
    X_std.columns = final_data.columns
```

```
[ ] #split the dataset into training and test datasets
    import numpy as np
    from sklearn.model_selection import train_test_split

    # Transform data into features and target
    X = np.array(my_data.iloc[:,1:16])
    y = np.array(my_data['y'])

    # split into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

```
[ ] # loading library
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import recall_score

    #Neighbors
    neighbors = np.arange(0,25)

    for k in neighbors:
        k_value = k+1
        knn = KNeighborsClassifier(n_neighbors = k_value)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        print(accuracy_score(y_test, y_pred))
```

Prediction and Accuracy Score

```
knn = KNeighborsClassifier(n_neighbors = 23)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
y_pred = knn.predict(X_test)

# evaluate accuracy
KNN=accuracy_score(y_test, y_pred)   #Accuracy of KNN model
print('Accuracy_score:',KNN)
```

```
Accuracy_score: 0.8977238239757208
```

Confusion Matrix

```
print('Confusion_matrix:')
print(metrics.confusion_matrix(y_test, y_pred))
```

```
Confusion_matrix:
[[5845    5]
 [ 669   71]]
```

Precision and Recall

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

```
print('Precision: %.3f' % precision_score(y_test, y_pred))
```

```
Precision: 0.934
```

```
print('Recall: %.3f' % recall_score(y_test, y_pred))
```

```
Recall: 0.096
```

Appendix 7

Other Models

Naive Bayes

```
[ ]  from sklearn import preprocessing
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import accuracy_score
```

```
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.30, random_state = 7)
```

```
[ ]  clf = GaussianNB()
     clf.fit(X_train,Y_train)
```

```
     GaussianNB()
```

```
[ ]  Y_pred = clf.predict(X_test)
```

Accuracy Score

```
[ ]  NB=accuracy_score(Y_test, Y_pred, normalize = True) #Accuracy of Naive Bayes' Model
     print('Accuracy_score:',NB)
```

```
     Accuracy_score: 1.0
```

Confusion Matrix

```
print('Confusion_matrix of NB:')
print(metrics.confusion_matrix(Y_test,Y_pred))
```

```
Confusion_matrix of NB:
[[8718    0]
 [   0 1167]]
```

Precision and Recall

```
[ ] print('Precision: %.3f' % precision_score(Y_test, Y_pred))
```

```
Precision: 1.000
```

```
[ ] print('Recall: %.3f' % recall_score(Y_test, Y_pred))
```

```
Recall: 1.000
```

```
[ ]
```

Appendix 7

Other Models

SVM

Support Vector Machine

```
[ ]  data=CD
```

```
[ ]  # split the datasets into training and test data
     X = data.drop('y', axis=1)
     y = data.y
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 555, test_size= 0.30)
```

```
[ ]  from sklearn import svm
     from matplotlib import pyplot as plt
     %matplotlib inline
```

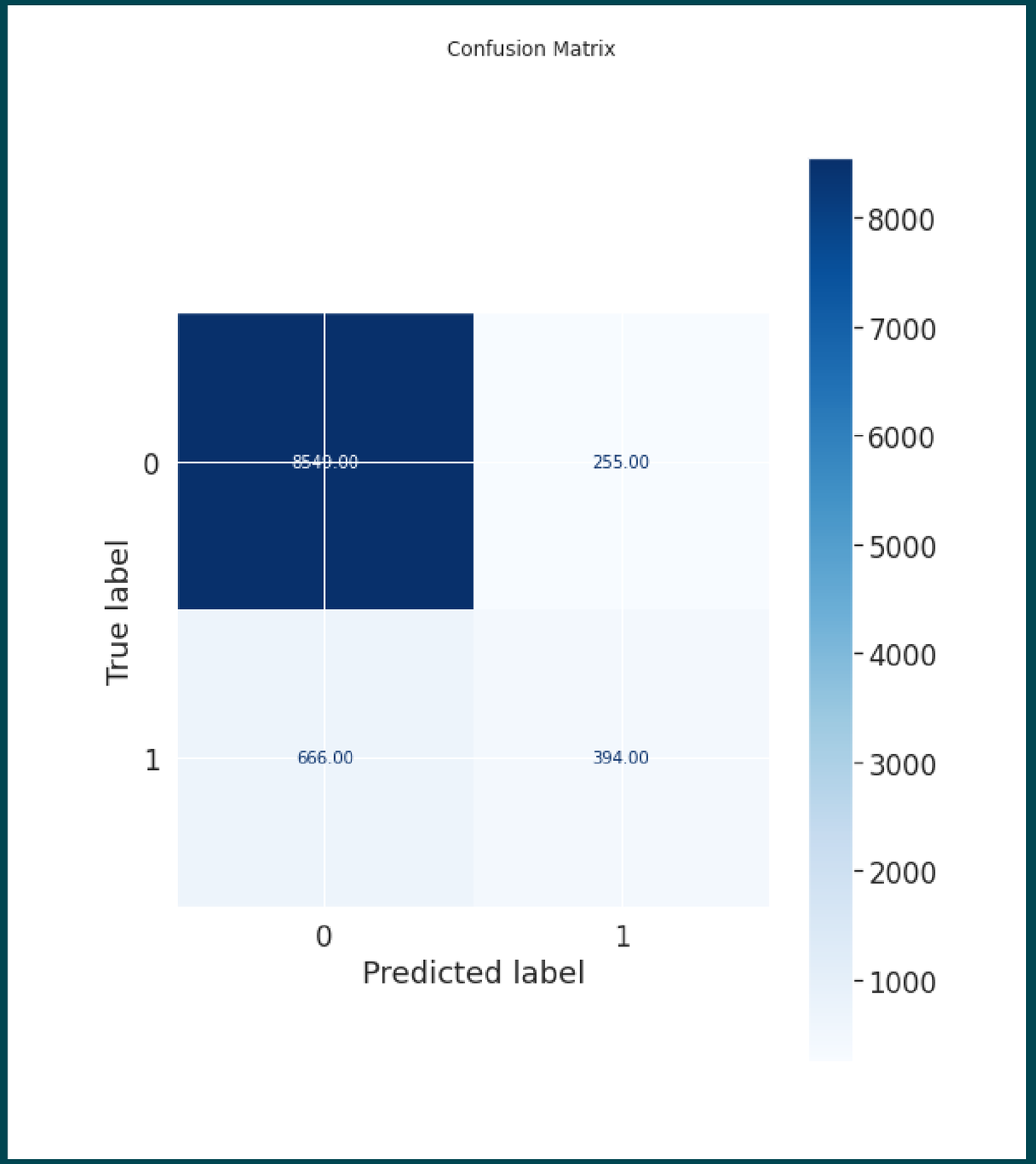
```
[ ]  svc = svm.SVC(kernel='rbf', C=70, gamma=0.001).fit(X_train,y_train)
     predictionsvm = svc.predict(X_test)
     from sklearn.metrics import classification_report
     print(classification_report(y_test,predictionsvm))
```

	precision	recall	f1-score	support
0	0.93	0.97	0.95	8804
1	0.61	0.37	0.46	1060
accuracy			0.91	9864
macro avg	0.77	0.67	0.70	9864
weighted avg	0.89	0.91	0.90	9864

```
[64]  predictionsvm = svc.predict(X_test)
      percentage = svc.score(X_test,y_test)
      percentage
```

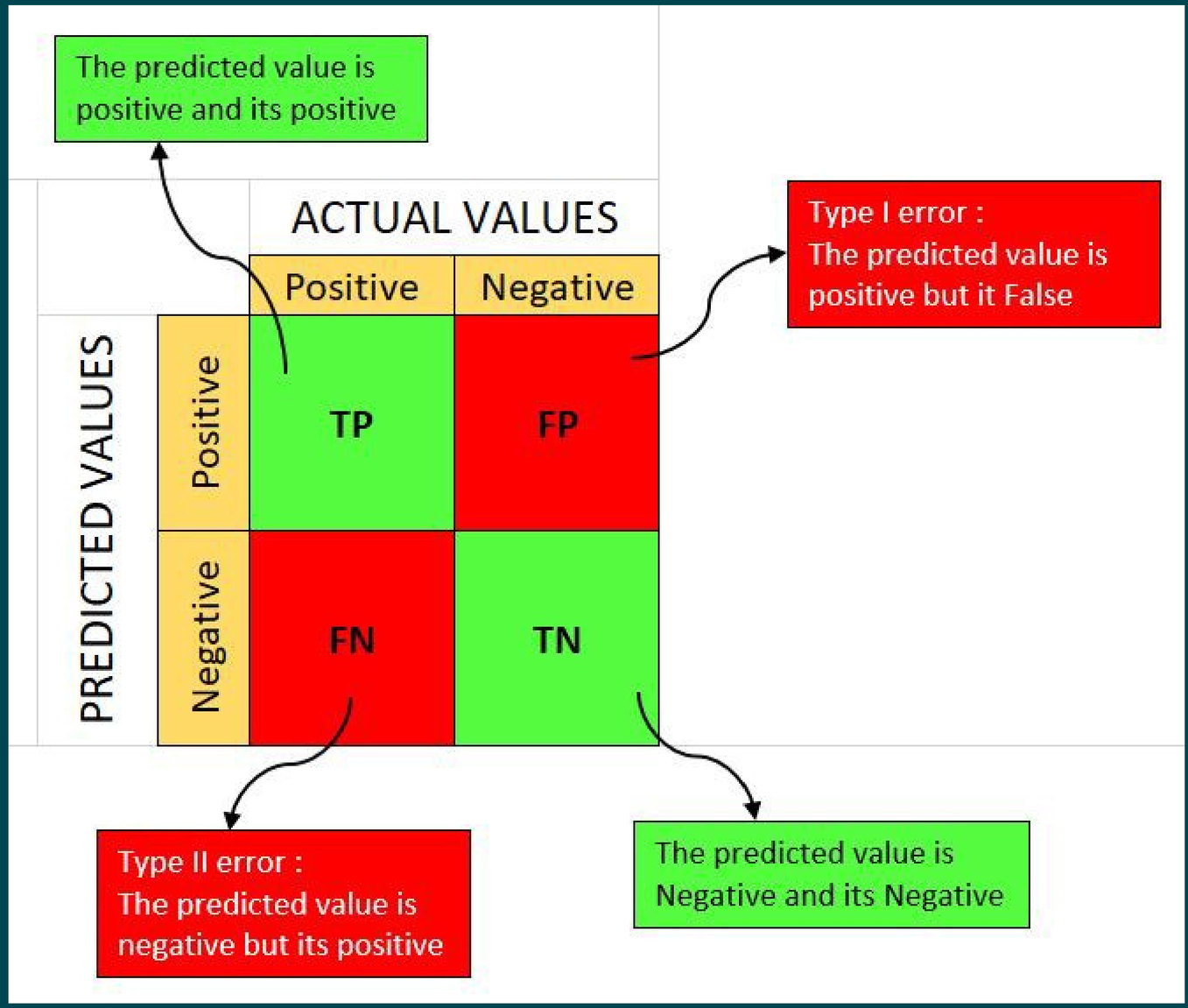
```
0.9066301703163017
```

```
from sklearn.metrics import plot_confusion_matrix
plt.rcParams["figure.figsize"] = (8, 10)
fig=plot_confusion_matrix(svc, X_test, y_test,display_labels=["0",'1'],cmap=plt.cm.Blues,values_format = '.2f')
fig.figure_.suptitle("Confusion Matrix ")
plt.show()
```


Confusion Matrix
8540.00
255.00
666.00
394.00
0
1
True label
0
1
Predicted label
8000
7000
6000
5000
4000
3000
2000
1000

What is a Confusion Matrix?

A confusion matrix visualizes and summarizes the performance of a classification algorithm.



CONFUSION MATRIX COMPARISON

Appendix 8

8578	217
680	389

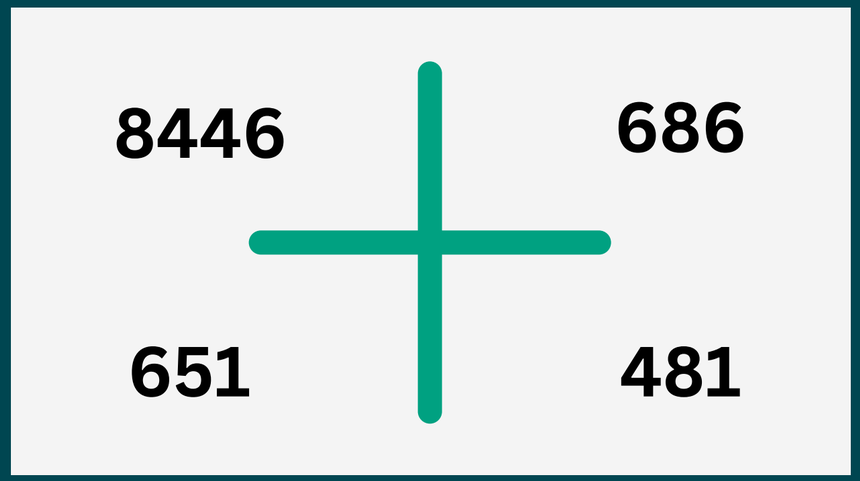
Logistic Regression

8497	307
743	317

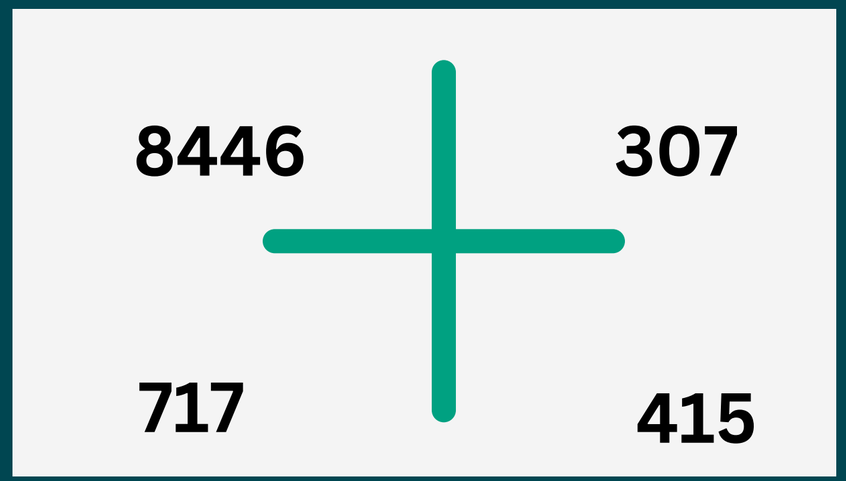
KNN

8256	462
846	321

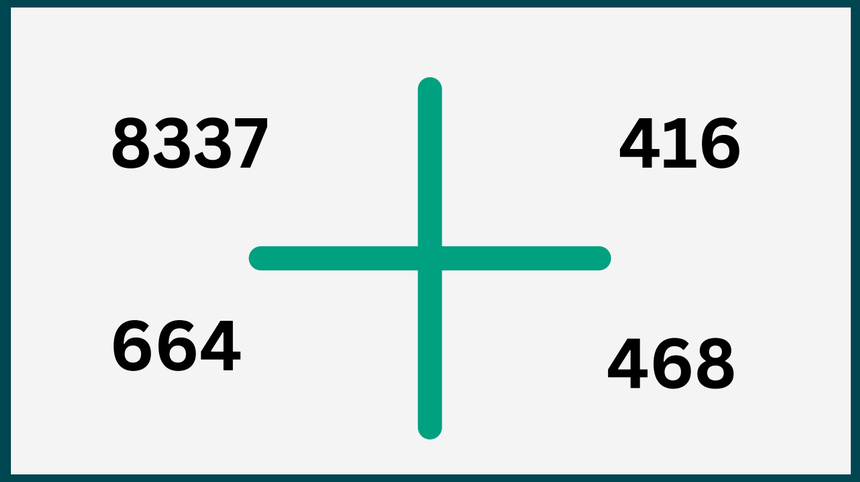
Naive Bayes



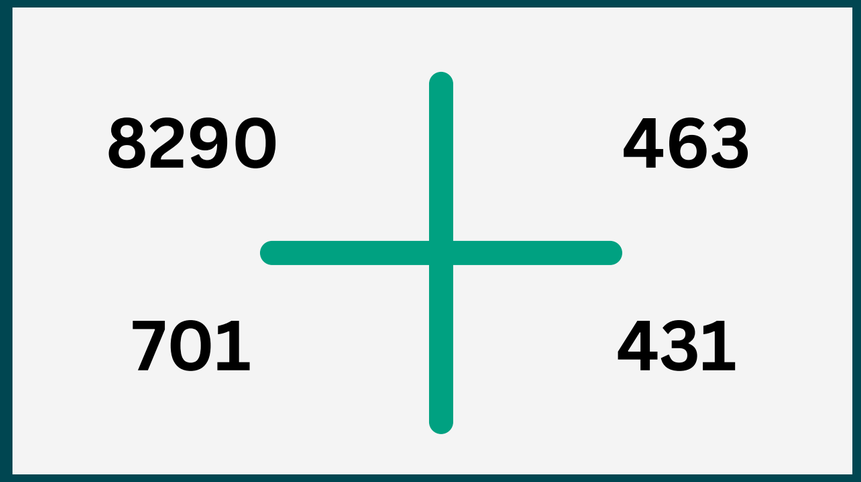
Decision Tree



Random Forest



Bagging



Boosting

MODEL EVALUATION

MODEL	SENSITIVITY	SPECIFICITY	PRECISION	F1 SCORE	Matthew's Correlation Coefficient
Logistic Regression	0.9266	0.6419	0.9753	0.9503	0.4391
KNN	0.9196	0.5080	0.9651	0.9418	0.3361
Naive Bayes	0.9071	0.4100	0.9470	0.9266	0.2653
Decision Tree	0.9284	0.4122	0.9249	0.9267	0.3452
Random Forest	0.9218	0.5748	0.9649	0.9428	0.4057
Bagging	0.9262	0.5294	0.9525	0.9392	0.4083
Boosting	0.9220	0.4821	0.9471	0.9344	0.3640
SVM	0.9277	0.6071	0.9710	0.9488	0.4281