Name : *Saurabh Mirchandani*
Roll No. : *20BCM073*
Subject : *Data Mining*
Subject Code : *CSI0803*
Assignment- 2
Date : *25-1-2023*

Assignment-2

Aim:- Data mining and Visualization

Banking Dataset - Marketing Targets

About Dataset

Content

The data is related to the 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 ('yes') or not ('no') subscribed by the customer or not. The data folder contains two datasets:-

- train.csv: 45,211 rows and 18 columns ordered by date (from May 2008 to November 2010)
- test.csv: 4521 rows and 18 columns with 10% of the examples (4521), randomly selected from train.csv

DESCRIPTION OF THE TASKS

TASK/S:

The classification goal is to predict if the client will subscribe to a term deposit.

DATASET:

The data is related to direct marketing campaigns (phone calls) 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 ('no') subscribed by the customer or not. The data folder contains two datasets:

- train.csv: 45,211 rows and 18 columns ordered by date (from May 2008 to November 2010)
- test.csv: 4,521 rows and 18 columns with 10% of the examples (4,521), randomly selected from train.csv

Specifically, these are the following features within the datasets:

1. age (numerical)
2. job: type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
3. marital: marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
4. education (categorical: "unknown", "secondary", "primary", "tertiary")
5. default: has credit in default? (categorical: "yes", "no")
6. balance: average yearly balance, in euros (numerical)
7. housing: has housing loan? (categorical: "yes", "no")
8. loan: has personal loan? (categorical: "yes", "no")
9. contact: contact communication type (categorical: "unknown", "telephone", "cellular")

10. day: last contact day of the month (numerical)
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; includes last contact (numerical)
14. pdays: number of days that passed by after the client was last contacted from a previous campaign; -1 means client was not previously contacted (numerical)
15. previous: number of contacts performed before this campaign and for this client (numerical)
16. poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success").
17. y **(TARGET)** - has the client subscribed a term deposit? (categorical: "yes", "no")

Some Overview Of importance of Data Visualization in banking sector

Data visualization is much more than PowerPoint presentations. It creates detailed graphs and charts to help employees understand large datasets in less time. There are several uses of data visualization in the banking industry.

Uses of Data Visualization in Finance:-

Easier Interpretation of data:-

Data visualization is an easy way to understand and learn from data. Since many of us are visual learners, using data visualization helps convey the message/ insight easily compared to blocks of text or spreadsheets.

Moreover, the traditional methods can only provide static data based on historical information. By the time the latest data is collected, cleaned, and processed, it becomes outdated and may not be as effective when used in decision-making. Banking professionals can take one look at the data visualization report of a customer's journey with the bank and determine the best financial service.

Quick Identification

There's no need to sort and filter countless rows and columns of data. You can generate graphical representations of any dataset. Create bar charts, line charts, pie charts, heat maps, etc., to instantly identify the macro and micro details of the same situation.

For example, you can look at the graph of monthly transactions in a bank branch to identify which month recorded the highest number and why.

Big Data is not a Problem

PowerPoint is not enough to create reports for big data. But Power BI and data visualization are perfect for big data. Since the financial industry deals with huge volumes of data, using data visualization tools is the best choice to understand datasets.

Detect Anomalies and patterns

Financial fraud is one of the biggest concerns in the banking sector. Data visualization reports help detect patterns you might otherwise not notice. The reports give employees the necessary information to minimize the risk of fraudulent transactions and prevent them from occurring. Banks and insurance companies have separate dashboards for fraud detection and risk management.

Data Sharing and collaboration

The visualization reports are not limited to only a few employees in the bank. The reports can be shared between departments, teams, and employees. This simplifies data sharing and collaboration within the bank. Teams from different branches can work together irrespective of their locations.

Data Analytics Project

For Banking Data

Classification with Python

In this notebook, I predict which customers are more likely to respond positively to a bank marketing call by setting up a regular savings deposit or subscribing the term "made_deposit".

Three classification algorithms will be developed in order to predict the target variable. Logistic Regression, Decision Tree and Multi-Layer Perceptron (MLP).

The analysis of the project includes Data Summary, Data Preparation, Modelling, Results and Errors using Evaluation Metrics, Confusion Matrices and ROC Curve.

Import libraries

In [4]:
```
import pandas as pd
import numpy as np
from scipy import stats

#import seaborn
import seaborn as sns
```

Read the data using pandas

In [5]:
```
df = pd.read_csv("/Users/stella/Google Drive/UNISTI/MSc Data Science for Business/Data
```

In [6]:
```
# A glance in the dataset

df.head()
```

Out[6]:

	accountID	town	country	age	job	married	education	defaulted?	current_balanc

0	24634684	Crawley	UK	39	management	married	tertiary	no	-19(
1	80795929	Southend-on-Sea	UK	53	housemaid	married	primary	no	36
2	30786087	Gillingham	UK	79	retired	divorced	primary	no	278
3	38925327	Gateshead	UK	43	services	single	secondary	no	55(
4	30560733	Eastbourne	UK	35	technician	single	secondary	no	34

5 rows × 21 columns

In [10]:
```
# Shape of the dataset (rows and columns)

df.shape
```

Out[10]: (8000, 21)

In [ ]:
```
df.columns
```

In [11]:
```
# The types of the variables

df.dtypes
```

Out[11]:
```
accountID                int64
town                    object
country                 object
age                      int64
job                     object
married                 object
education               object
defaulted?              object
current_balance          int64
housing                 object
has_loan                object
last_contact            object
cc_tr                    int64
last_contact_day         int64
last_contact_month      object
last_contact_duration_s  int64
campaign                 int64
days_since_last_contact  int64
previous                 int64
poutcome                object
made_deposit            object
dtype: object
```

Null or missing values in the dataset

In [12]:
```
missing_data = df.isnull()
missing_data.head(5)
```

Out[12]:

	accountID	town	country	age	job	married	education	defaulted?	current_balance	housing
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False

5 rows × 21 columns

In [13]:
```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
accountID
False    8000
Name: accountID, dtype: int64

town
False    8000
Name: town, dtype: int64

country
False    8000
Name: country, dtype: int64

age
False    8000
Name: age, dtype: int64

job
False    8000
Name: job, dtype: int64

married
False    8000
Name: married, dtype: int64

education
False    8000
Name: education, dtype: int64

defaulted?
False    8000
Name: defaulted?, dtype: int64

current_balance
False    8000
Name: current_balance, dtype: int64

housing
False    8000
Name: housing, dtype: int64

has_loan
False    8000
Name: has_loan, dtype: int64

last_contact
False    8000
Name: last_contact, dtype: int64

cc_tr
False    8000
Name: cc_tr, dtype: int64

last_contact_day
False    8000
Name: last_contact_day, dtype: int64

last_contact_month
```

```
False    8000
Name: last_contact_month, dtype: int64

last_contact_duration_s
False    8000
Name: last_contact_duration_s, dtype: int64

campaign
False    8000
Name: campaign, dtype: int64

days_since_last_contact
False    8000
Name: days_since_last_contact, dtype: int64

previous
False    8000
Name: previous, dtype: int64

poutcome
False    8000
Name: poutcome, dtype: int64

made_deposit
False    8000
Name: made_deposit, dtype: int64
```

Limiting floats to three decimal points

In [14]:
```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

Descriptive Statistics of the numerical variables

In [15]:
```
df.describe()
```

Out[15]:

	accountID	age	current_balance	cc_tr	last_contact_day	last_contact_duration_s	ca
count	8000.000	8000.000	8000.000	8000.000	8000.000	8000.000	8
mean	55407459.576	41.212	1516.521	3.011	15.709	373.484	
std	25961372.817	11.973	3168.518	1.420	8.433	349.518	
min	10000398.000	18.000	-3058.000	1.000	1.000	-60.000	
25%	33058255.500	32.000	120.000	2.000	8.000	139.000	
50%	55587795.000	39.000	545.500	3.000	15.000	256.000	
75%	77690442.500	49.000	1694.000	4.000	22.000	502.250	
max	99995994.000	93.000	81204.000	5.000	31.000	3881.000	

Correlation Matrix of all numerical variables

In [16]:
```
df.corr()
```

Out[16]:

	accountID	age	current_balance	cc_tr	last_contact_day	last_contact_durati
accountID	1.000	0.005	0.017	0.006	0.013	
age	0.005	1.000	0.118	0.006	-0.002	
current_balance	0.017	0.118	1.000	0.028	0.018	
cc_tr	0.006	0.006	0.028	1.000	0.007	
last_contact_day	0.013	-0.002	0.018	0.007	1.000	-(
last_contact_duration_s	0.013	0.006	0.022	0.006	-0.012	
campaign	-0.006	-0.016	-0.010	-0.010	0.135	-(
days_since_last_contact	-0.010	0.007	0.024	-0.013	-0.071	-(
previous	-0.008	0.022	0.033	-0.018	-0.056	-(

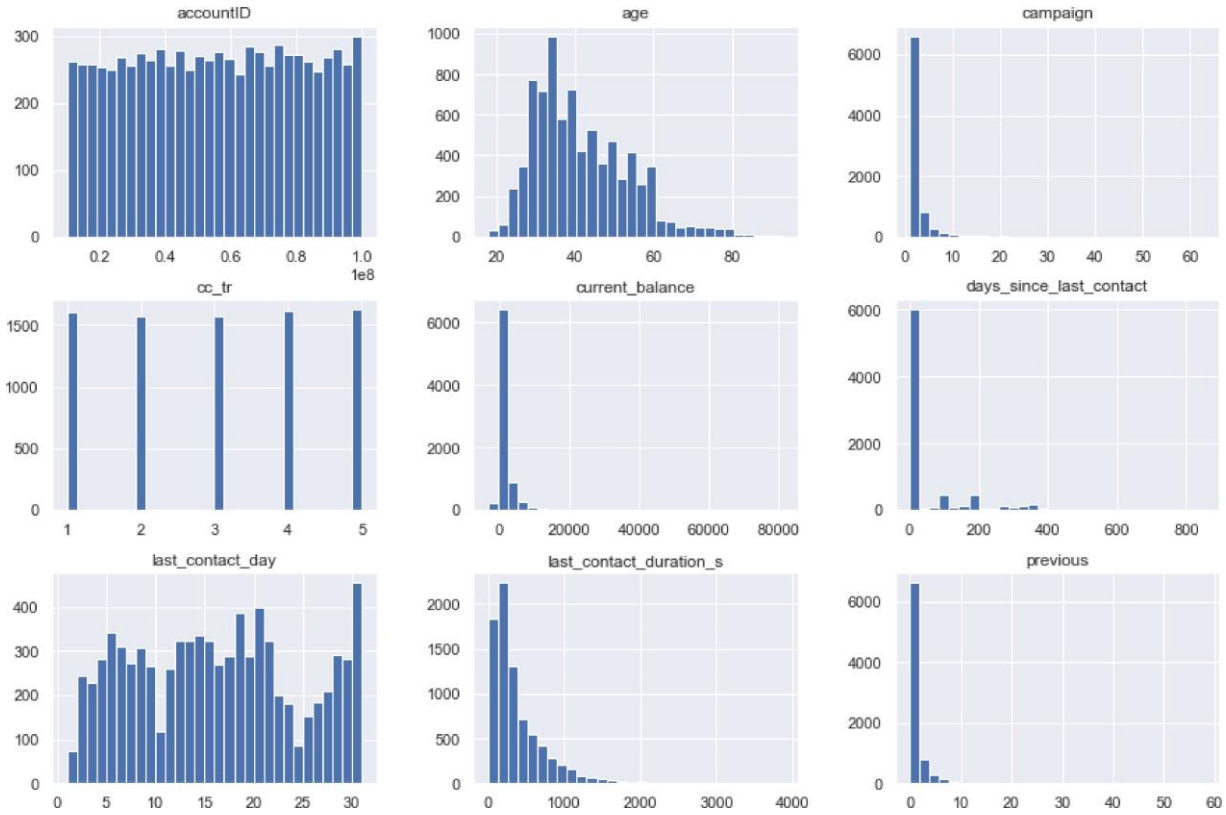
In [ ]:
```
# The independent variable "made_deposit" : yes or no (binary)

# How many of each class is found in the data set

df['made_deposit'].value_counts()
```

In [ ]:
```
df.groupby('made_deposit').mean()
```

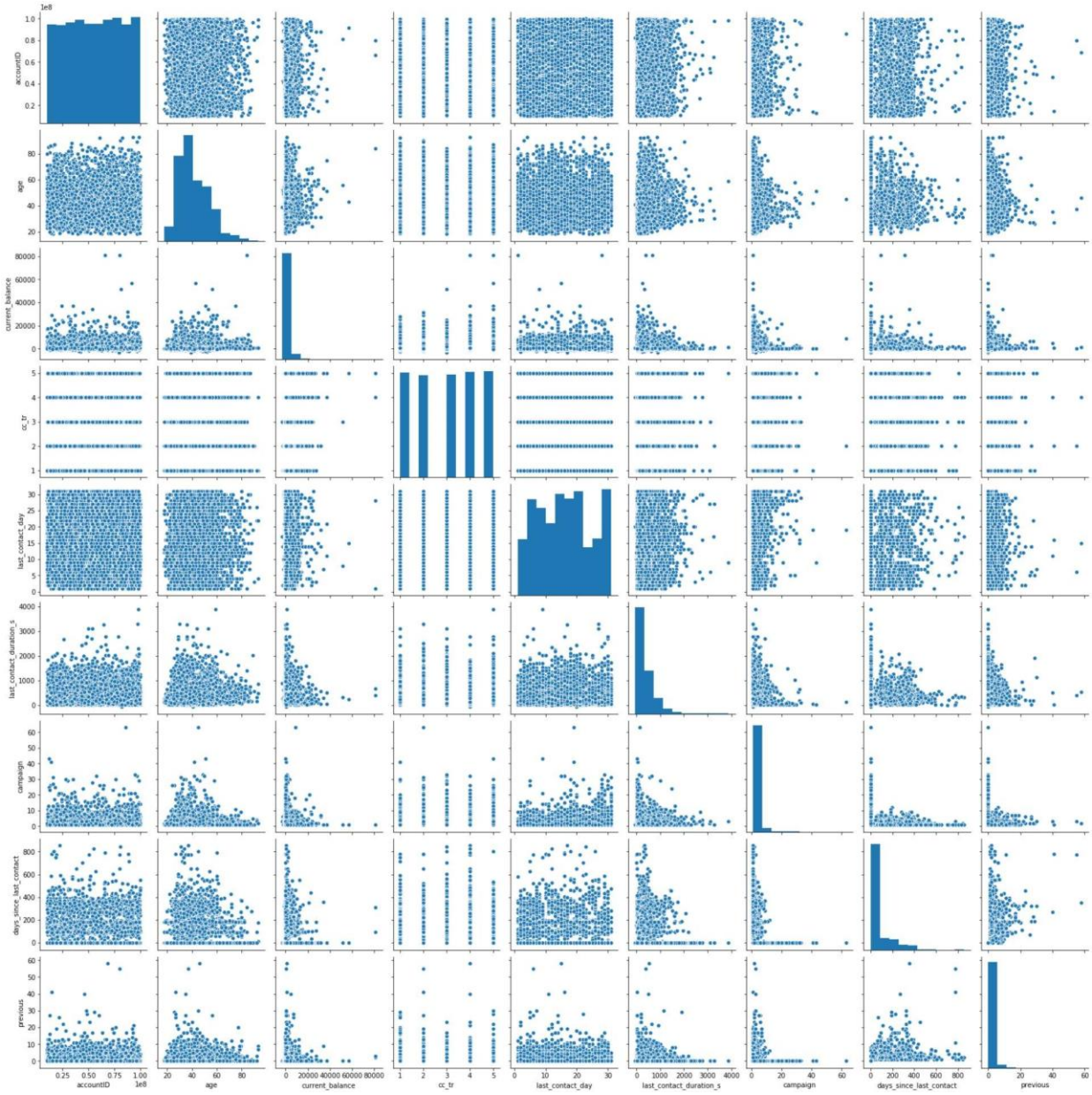
Data Visualisation

Numerical Variables

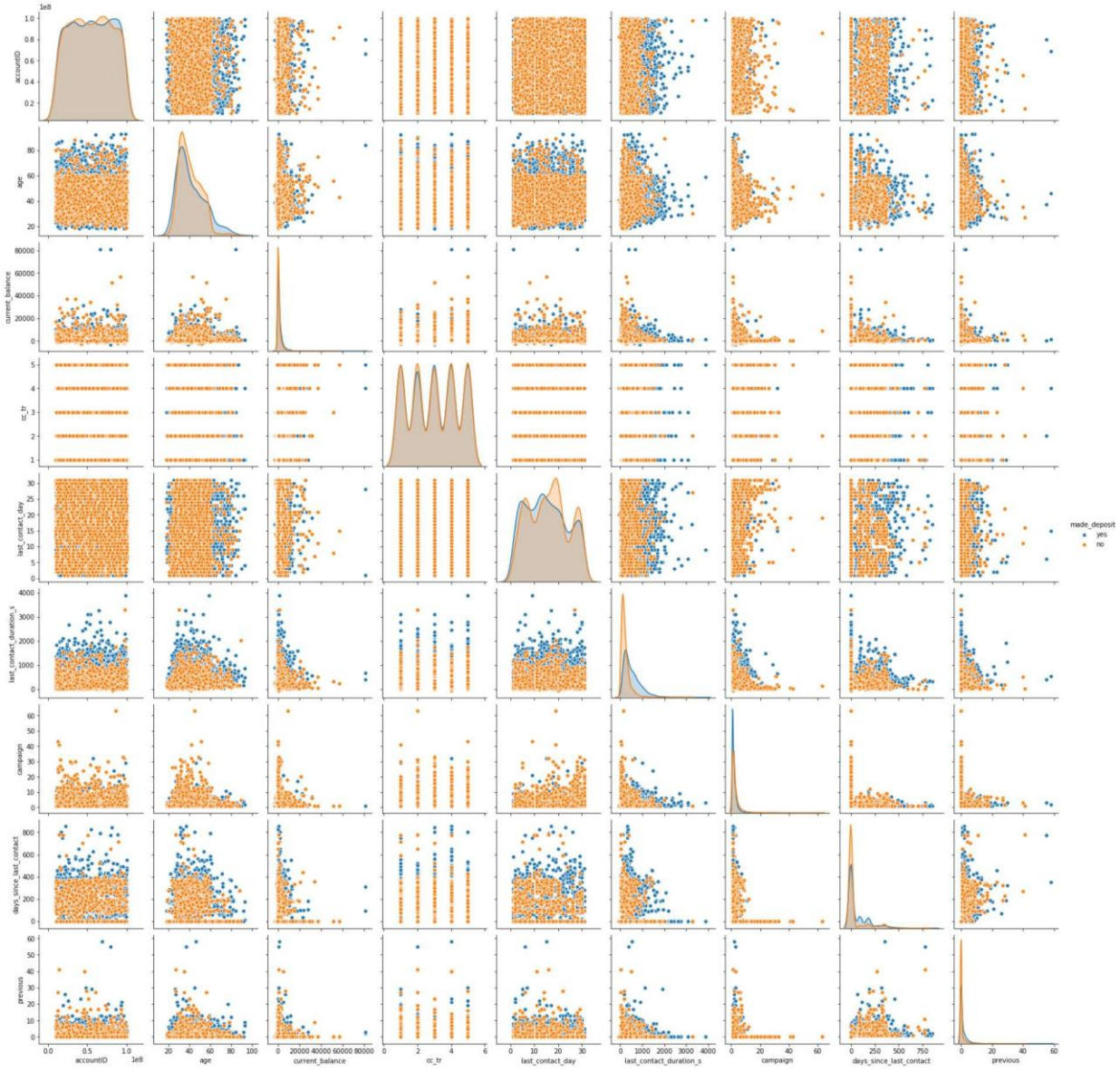
In [81]:
```
#import matplotlib library
%matplotlib inline
import matplotlib.pyplot as plt


# Histogram for each numerical attribute
df.hist(bins=30, figsize=(15,10))
plt.show()
```



In [21]:
```
# Pairs plot

sns.pairplot(df);
```



In [22]:
```
# Using seaborn library for data visualisation with pairs plot
# All numerical variables' plots are shown, grouped by the feature of interest - deper

sns.pairplot(df, hue="made_deposit");
```



Categorical Variables (Values and Visualisation)

This will reveal data entry errors and they will be fixed in the Data Preparation section of the notebook

In [23]:
```
df['town'].value_counts()
```

Out[23]:
```
London          472
Birmingham      353
Glasgow         335
Bristol         260
Liverpool       233
                ...
Dundee           52
Crawley          52
Aberdeen         52
Exeter           50
Bradford         46
Name: town, Length: 101, dtype: int64
```

In [24]:
```
# We see branches operating not only in UK but also in France, Germany ...
df['country'].value_counts()
```

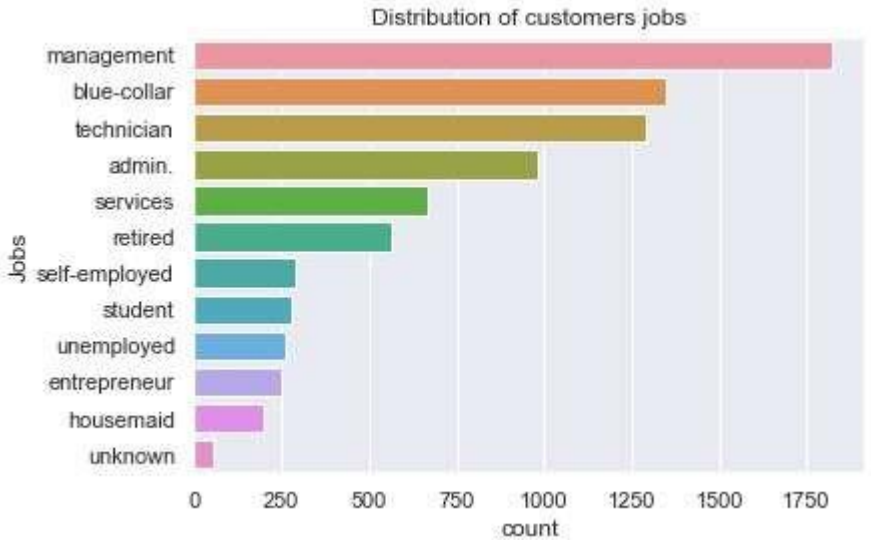
Out[24]:
```
UK          7995
France         2
Germany        1
USA            1
Portugal       1
Name: country, dtype: int64
```

In [25]:
```
df['job'].value_counts()
```

Out[25]:
```
management       1824
blue-collar      1348
technician       1292
admin.            979
services          666
retired           562
self-employed     292
student           277
unemployed        259
entrepreneur      248
housemaid         199
unknown            54
Name: job, dtype: int64
```

In [26]:
```
# Visualising the 'job' categories of the customers

sns.set(style="darkgrid")

ax = sns.countplot(y="job", data=df, order=['management', 'blue-collar', 'technician',
ax.set_title('Distribution of customers jobs')
ax.set_ylabel('Jobs')
```

Out[26]:
```
Text(0, 0.5, 'Jobs')
```

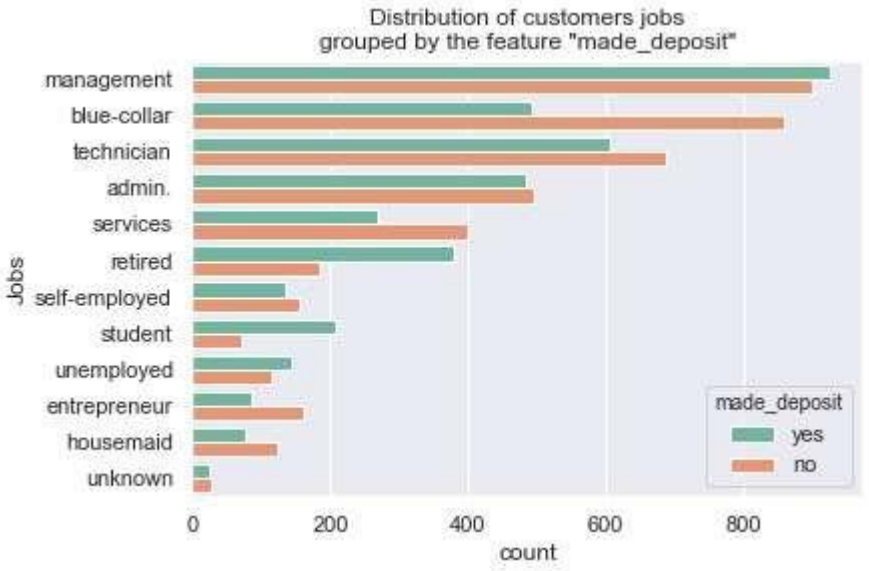


In [52]:
```
df.groupby(['job'])['made_deposit'].value_counts(normalize=True)
```

Out[52]:
```
job            made_deposit
admin.         no             0.506
               yes            0.494
blue-collar    no             0.636
               yes            0.364
entrepreneur   no             0.649
               yes            0.351
housemaid      no             0.613
               yes            0.387
management     yes            0.507
               no             0.493
retired        yes            0.673
               no             0.327
self-employed  no             0.534
               yes            0.466
services       no             0.598
               yes            0.402
student        yes            0.747
               no             0.253
technician     no             0.531
               yes            0.469
unemployed     yes            0.552
               no             0.448
unknown        no             0.519
               yes            0.481
Name: made_deposit, dtype: float64
```

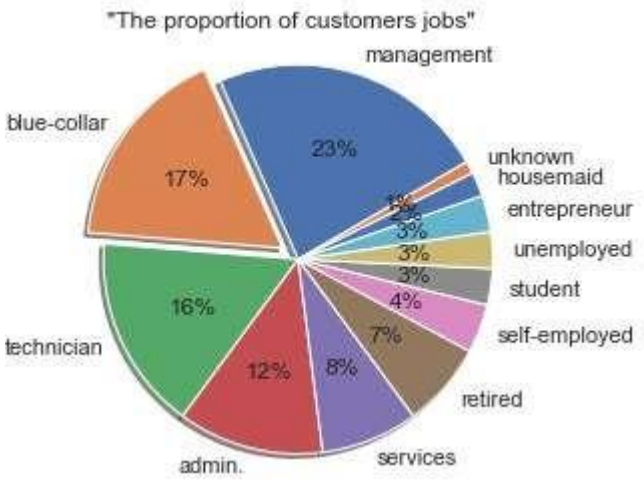
In [53]:
```
# Visualising the 'job' categories of the customers grouped by made_deposit

sns.set(style="darkgrid")

ax = sns.countplot(y="job", hue="made_deposit", data=df, palette="Set2", order=['manag
ax.set_title('Distribution of customers jobs\ngrouped by the feature "made_deposit"')
ax.set_ylabel('Jobs')
```

Out[53]:
```
Text(0, 0.5, 'Jobs')
```



Or, visualising the 'job' categories of the customers using a Pie Chart.

In [54]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:

labels = 'management', 'blue-collar', 'technician', 'admin.', 'services', 'retired', '
sizes = [23, 17, 16, 12, 8, 7, 4, 3, 3, 3, 2, 1]
explode = (0, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)  # only "explode" the 2nd slice (i.e.

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.0f%%', frame=False,
        shadow=True, startangle=30)  #radius=30
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title('"The proportion of customers jobs"')
plt.show()
```

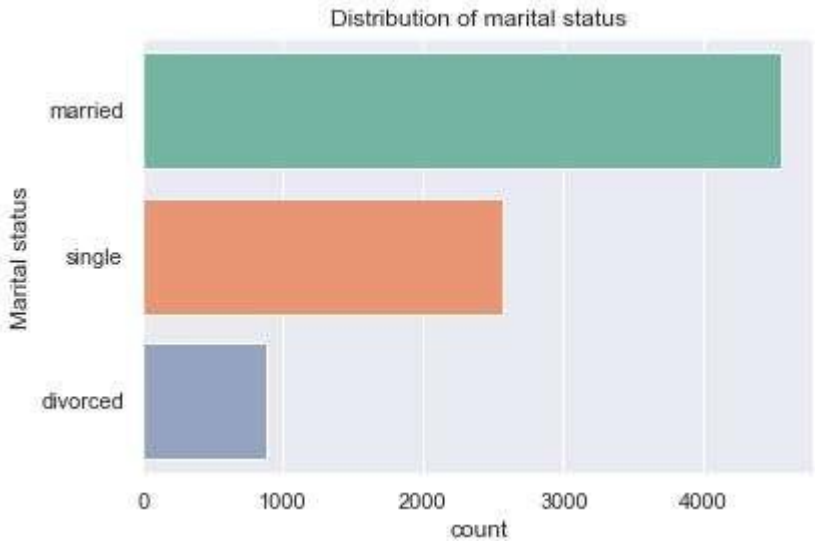


In [55]:
```
df['married'].value_counts()
```

Out[55]:
```
married     4557
single      2559
divorced     884
Name: married, dtype: int64
```

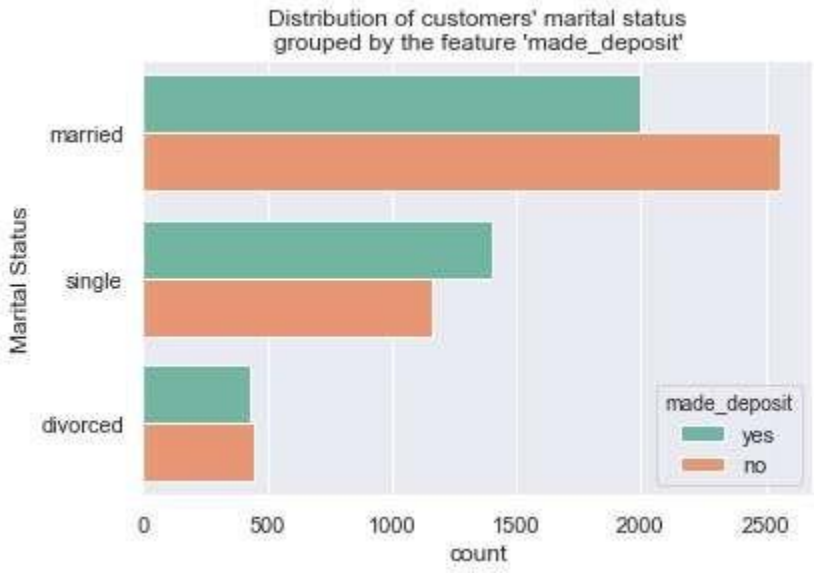
In [56]:
```
# Visualising the marital status of the customers

sns.set(style="darkgrid")

ax = sns.countplot(y="married", data=df, palette="Set2", order=['married', 'single', '
ax.set_title("Distribution of marital status")
ax.set_ylabel('Marital status')
```

Out[56]:
```
Text(0, 0.5, 'Marital status')
```



In [57]:
```
# Visualising the marital status of the customers grouped by made_deposit
sns.set(style="darkgrid")

ax = sns.countplot(y="married", hue="made_deposit", data=df, palette="Set2", order=['m
ax.set_title("Distribution of customers' marital status\ngrouped by the feature 'made_
ax.set_ylabel('Marital Status')
```

Out[57]:
```
Text(0, 0.5, 'Marital Status')
```



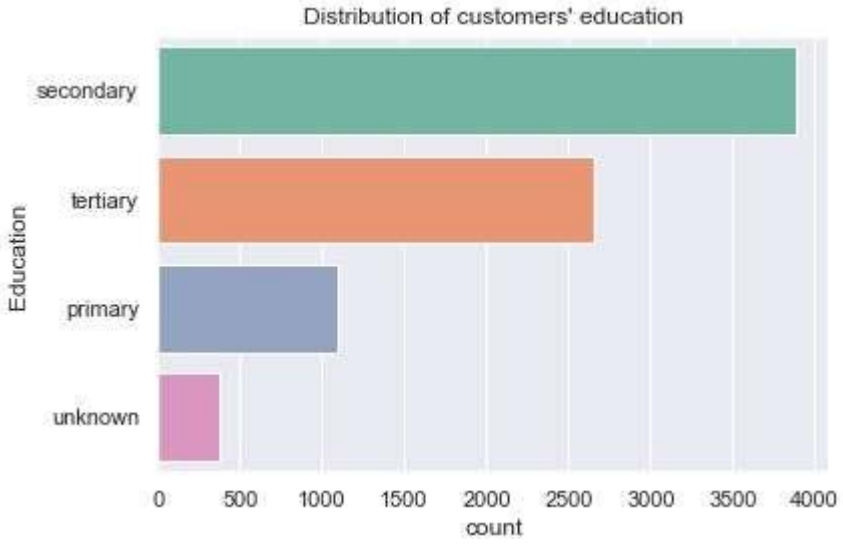
In [58]:
```
df['education'].value_counts()
```

Out[58]:
```
secondary    3888
tertiary     2653
primary      1088
unknown       371
Name: education, dtype: int64
```

In [32]:
```
# Visualising the 'education' of the customers

sns.set(style="darkgrid")

ax = sns.countplot(y="education", data=df, palette="Set2", order=['secondary', 'tertia
```

```
ax.set_title("Distribution of customers' education")
ax.set_ylabel('Education')
```

Out[32]: Text(0, 0.5, 'Education')

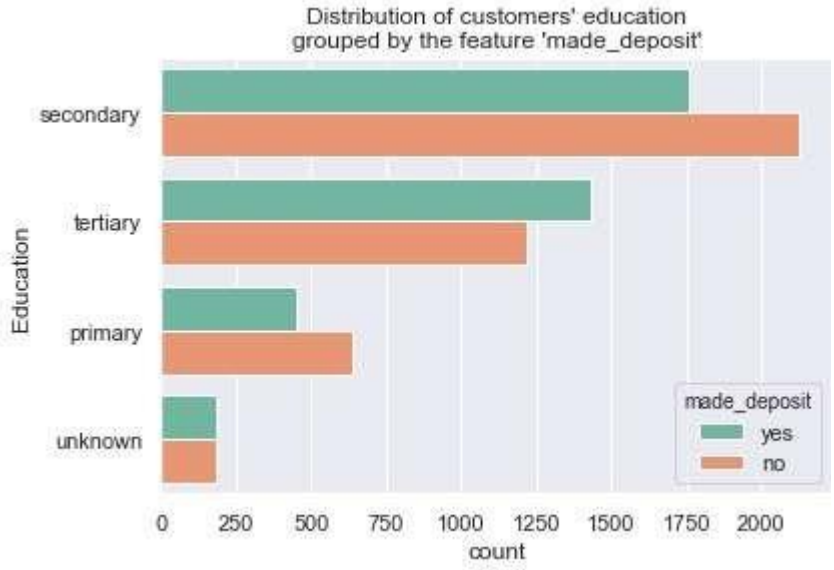


In [59]:
```
df.groupby(['education'])['made_deposit'].value_counts(normalize=True)
```

Out[59]:
```
education  made_deposit
primary    no             0.587
           yes            0.413
secondary  no             0.547
           yes            0.453
tertiary   yes            0.541
           no             0.459
unknown    no             0.504
           yes            0.496
Name: made_deposit, dtype: float64
```

In [60]:
```
# Visualising the 'education' of the customers grouped by made_deposit

sns.set(style="darkgrid")

ax = sns.countplot(y="education", hue="made_deposit", data=df, palette="Set2", order=[
ax.set_title("Distribution of customers' education\ngrouped by the feature 'made_depos
ax.set_ylabel('Education')
```

Out[60]: Text(0, 0.5, 'Education')

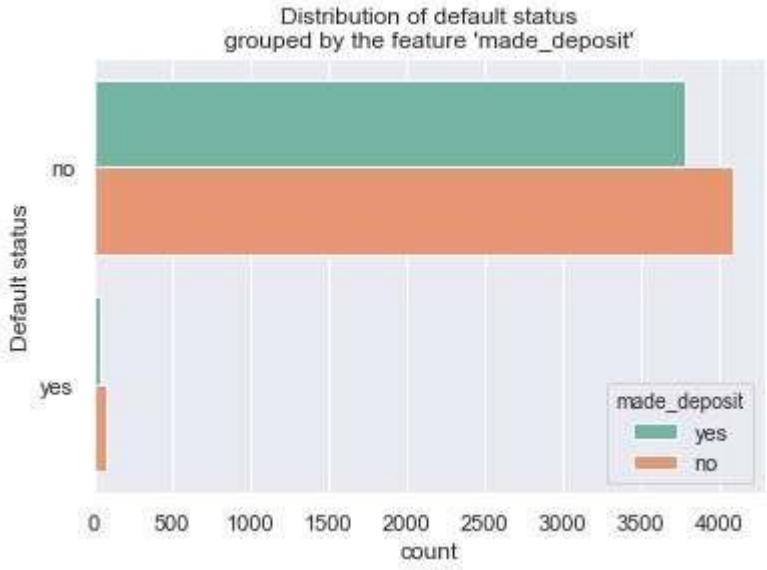


In [61]:
```
df['defaulted?'].value_counts()
```

Out[61]:
```
no     7880
yes     120
Name: defaulted?, dtype: int64
```

In [62]:
```
# Visualisation of the 'defaulted?'' status of the customers grouped by made_deposit

sns.set(style="darkgrid")

ax = sns.countplot(y="defaulted?", hue="made_deposit", data=df, palette="Set2", order=
ax.set_title("Distribution of default status\ngrouped by the feature 'made_deposit'")
ax.set_ylabel('Default status')
```

Out[62]:
```
Text(0, 0.5, 'Default status')
```



In [63]:
```
df['housing'].value_counts()
```

Out[63]:
```
no     4227
yes    3773
Name: housing, dtype: int64
```

In [64]:
```
df['has_loan'].value_counts()
```

Out[64]:
```
no     6964
yes    1031
n         5
Name: has_loan, dtype: int64
```

In [65]:
```
df['last_contact'].value_counts()
```

Out[65]:
```
cellular     5740
unknown      1693
telephone     565
cell            2
Name: last_contact, dtype: int64
```

In [66]:
```
df['last_contact_month'].value_counts()
```

Out[66]:
```
may    2037
jul    1087
aug    1064
jun     863
nov     669
apr     660
feb     552
oct     302
jan     255
sep     240
mar     194
dec      76
j         1
Name: last_contact_month, dtype: int64
```

In [67]:
```
df['poutcome'].value_counts()
```

Out[67]:
```
unknown    5960
failure     888
success     775
other       377
Name: poutcome, dtype: int64
```

Data Preparation

I will transform some of the variables fixing the data entry errors that have been revealed previously.

last_contact_month

In [68]:
```
# Replacing 'j' with the most frequent value between the months that start from j

df['last_contact_month'].replace('j', 'jul', inplace=True)
```

In [69]:
```
df.groupby(['last_contact_month'])['made_deposit'].value_counts(normalize=True)
```

Out[69]:
```
last_contact_month  made_deposit
apr                 yes            0.641
                    no             0.359
aug                 no             0.550
                    yes            0.450
dec                 yes            0.934
                    no             0.066
feb                 yes            0.591
                    no             0.409
jan                 no             0.584
                    yes            0.416
jul                 no             0.588
                    yes            0.412
jun                 no             0.547
                    yes            0.453
mar                 yes            0.907
                    no             0.093
may                 no             0.668
                    yes            0.332
nov                 no             0.575
                    yes            0.425
oct                 yes            0.818
                    no             0.182
sep                 yes            0.838
                    no             0.163
Name: made_deposit, dtype: float64
```

has_loan

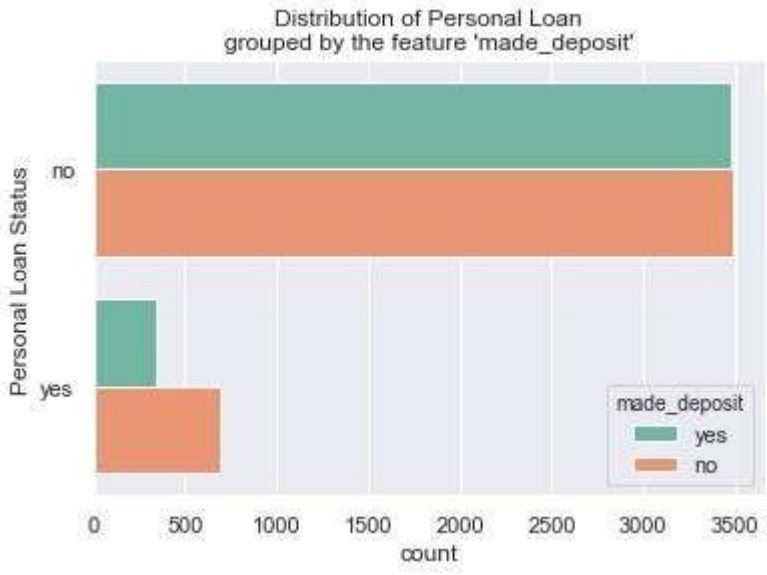
In [70]:
```
# Replacing all n with no ; 'has_loan'

df['has_loan'].replace('n', 'no', inplace=True)
```

In [71]:
```
df.groupby(['has_loan'])['made_deposit'].value_counts(normalize=True)
```

Out[71]:
```
has_loan  made_deposit
no        no             0.500
          yes            0.500
yes       no             0.666
          yes            0.334
Name: made_deposit, dtype: float64
```

In [72]:
```
# Visualisation of the 'has_loan' (meaning personal loan) status of the customers grou

sns.set(style="darkgrid")

ax = sns.countplot(y="has_loan", hue="made_deposit", data=df, palette="Set2", order=['
ax.set_title("Distribution of Personal Loan\ngrouped by the feature 'made_deposit'")
ax.set_ylabel('Personal Loan Status')
```

Out[72]:
```
Text(0, 0.5, 'Personal Loan Status')
```



last_contact_duration_s

The min value of 'last_contact_duration_s' is -60 seconds (refer to descriptive statistics table). It doesn't make sense to have a negative value as count of the seconds. This will be replaced with 60.

In [73]:
```
# Replacing '-60 ' with 60,assume that is 60, instead of the mean which is; : 373.4843

df['last_contact_duration_s'].replace(-60, 60, inplace=True)
```

In [75]:
```
print("The min last_contact_duration_s value is:", df['last_contact_duration_s'].min()
```

```
The min last_contact_duration_s value is: 2
```

days_since_last_contact

In [76]:
```
# Replacing '-1 ' with 1. Assuming that is 1, instead of the mean which is equal to 37

df['days_since_last_contact'].replace(-1, 1, inplace=True)
```

In [77]:
```
print("min days_since_last_contact value:", df['days_since_last_contact'].min())
```

```
min days_since_last_contact value: 1
```

last_contact

In [78]:
```
# Replacing 'cell' with 'cellular' (most frequaent value)
df['last_contact'].replace( 'cell', 'cellular', inplace=True)
```

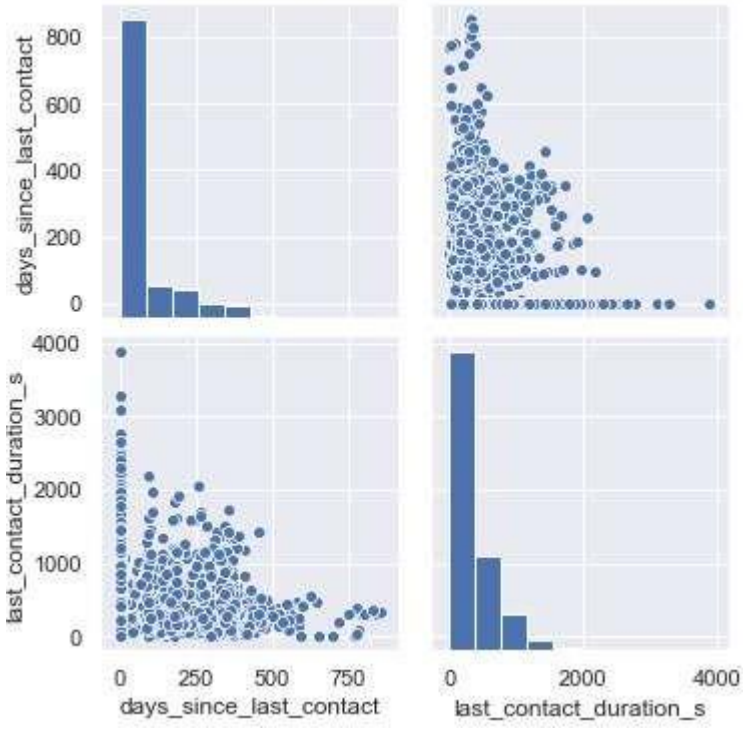
In [79]:
```
df.groupby(['last_contact'])['made_deposit'].value_counts(normalize=True)
```

Out[79]:
```
last_contact  made_deposit
cellular      yes            0.548
              no             0.452
telephone     yes            0.503
              no             0.497
unknown       no             0.765
              yes            0.235
Name: made_deposit, dtype: float64
```

Visualisation after the Transformation of some of the Variables

This will include Pairs Plot of two numerical variables (which have been transformed - corrected data entry errors) and Box Plots.

In [80]:
```
# Pairs plot of two numerical variables after correcting data entry errors

num_feat_after_trans = df[['days_since_last_contact', 'last_contact_duration_s']]
sns.pairplot(num_feat_after_trans);
```

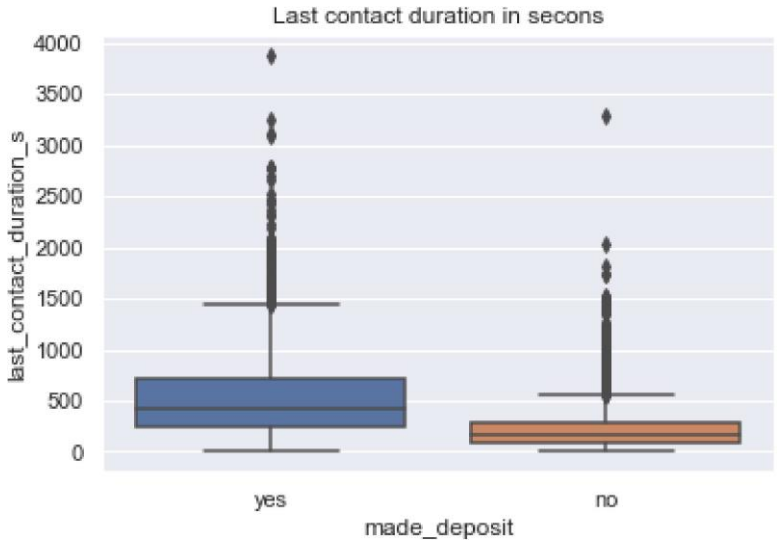


Outliers

Looking for outliers with the help of boxplots

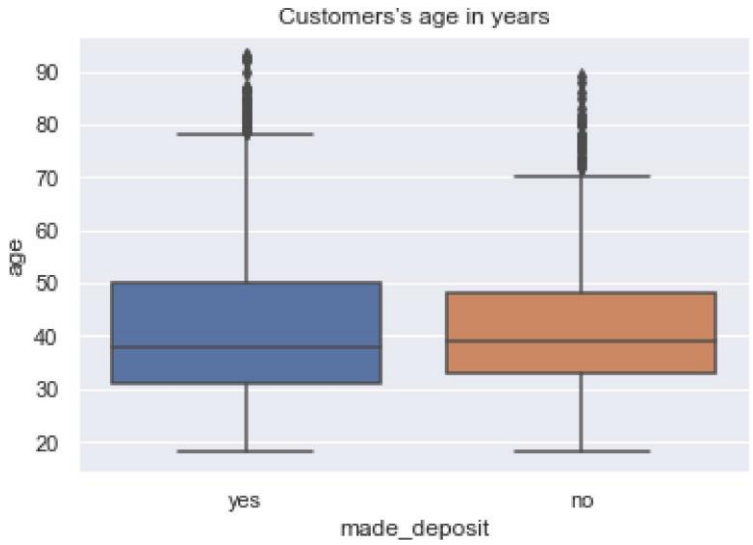
In [88]:
```
sns.boxplot(x="made_deposit", y="last_contact_duration_s", data=df).set_title('Last cc
```

Out[88]:
```
Text(0.5, 1.0, 'Last contact duration in secons')
```



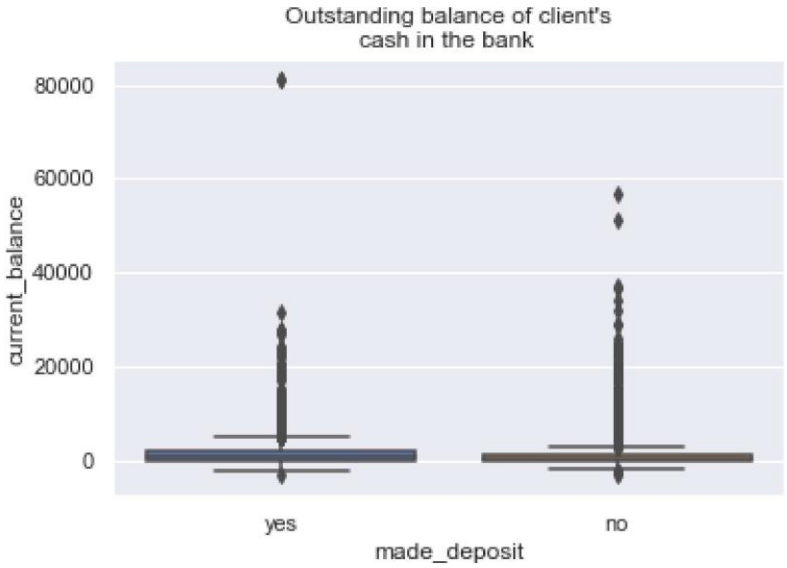
In [92]:
```
sns.boxplot(x="made_deposit", y="age", data=df).set_title("Customers's age in years")
```

Out[92]:
```
Text(0.5, 1.0, 'Customers's age in years')
```



In [93]:
```
sns.boxplot(x="made_deposit", y="current_balance", data=df).set_title("Outstanding bal
```

Out[93]:
```
Text(0.5, 1.0, "Outstanding balance of client's\ncash in the bank ")
```

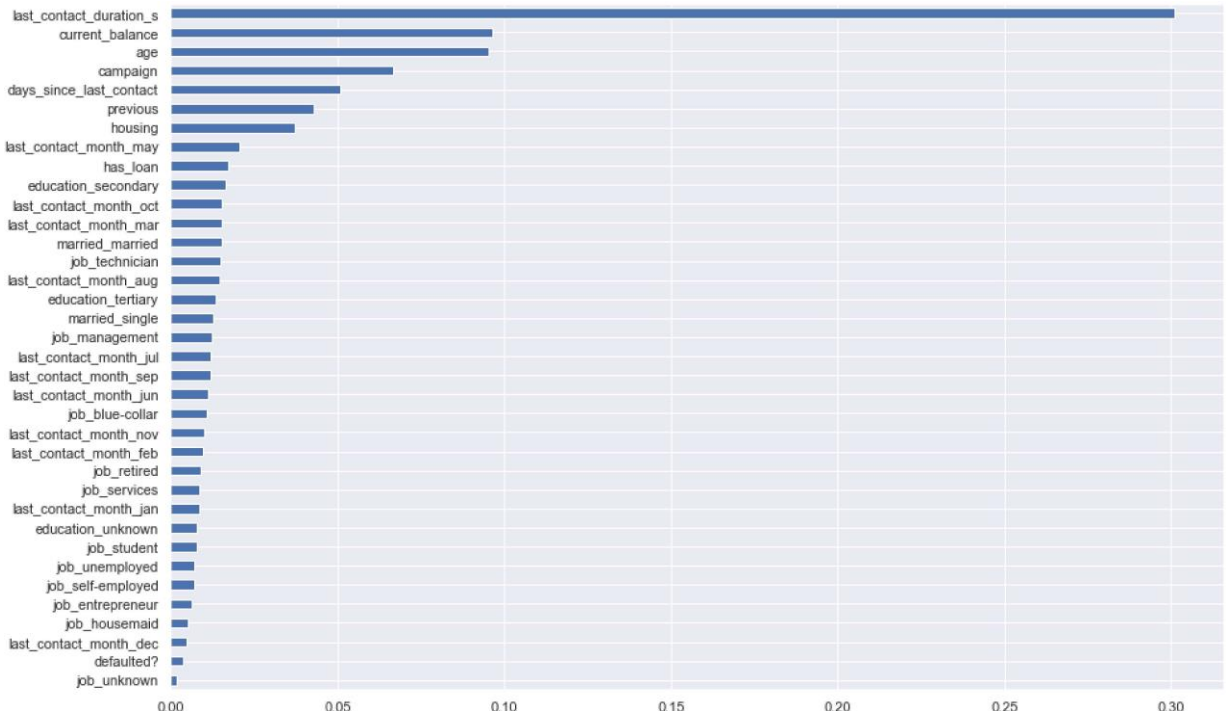


Normalize Data - Feature Scaling

In [125…

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel


#X, y defined previously in feature scaling
print("Original shape: " + str(X.shape))

# fit a tree-based model to the data
et_model = ExtraTreesClassifier(n_estimators=50)
et_model = et_model.fit(X, y)

# extract feature importances
feat_importances = et_model.feature_importances_
print("Importances: " + str(feat_importances))

# this is how the importances are used to select the features in the data
# in this case, it will use the mean importance as a threshold for selection
model = SelectFromModel(et_model, prefit=True)
X_new = model.transform(X)
print("Updated shape: " + str(X_new.shape))

# I plot the importances graphically...
# for this we match the importances with the names of the features
# in a pandas series
feat_ranks = pd.Series(feat_importances, index=featureNames)
feat_ranks.sort_values(inplace=True)
print("Ranks: ")
print(feat_ranks) # as text
plot=feat_ranks.plot(kind="barh", figsize=(15,10)) # as plot
```

```
Original shape: (8000, 36)
Importances: [0.0953049  0.00349806 0.09648106 0.03703422 0.01708778 0.30089838
 0.06669253 0.05097245 0.04269213 0.01086278 0.00609658 0.00523223
 0.01241622 0.00874639 0.00688191 0.00864548 0.00762297 0.01476698
 0.00699715 0.00179434 0.0151294  0.0127537  0.01629557 0.01347059
 0.0078073  0.01464524 0.00491946 0.00974169 0.00834553 0.01209065
 0.01124016 0.01517039 0.02058923 0.01009637 0.01522157 0.01175861]
Updated shape: (8000, 7)
Ranks:
job_unknown              0.002
defaulted?               0.003
last_contact_month_dec   0.005
job_housemaid            0.005
job_entrepreneur         0.006
job_self-employed        0.007
job_unemployed           0.007
job_student              0.008
education_unknown        0.008
last_contact_month_jan   0.008
job_services             0.009
job_retired              0.009
last_contact_month_feb   0.010
last_contact_month_nov   0.010
job_blue-collar          0.011
last_contact_month_jun   0.011
last_contact_month_sep   0.012
last_contact_month_jul   0.012
job_management           0.012
married_single           0.013
education_tertiary       0.013
last_contact_month_aug   0.015
job_technician           0.015
married_married          0.015
last_contact_month_mar   0.015
last_contact_month_oct   0.015
education_secondary      0.016
has_loan                 0.017
last_contact_month_may   0.021
housing                  0.037
previous                 0.043
days_since_last_contact  0.051
campaign                 0.067
age                      0.095
current_balance          0.096
last_contact_duration_s  0.301
dtype: float64
```



Also, I normalize the dataset (Feature X)

In [134…
```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[134]:
```
array([[ 1.48649391, -0.48014277, -0.35441529,  1.98661796, -0.94477258,
        -0.18474671, -0.53862006],
       [ 2.27623755, -0.48014277, -0.35441529, -0.17946936, -0.94477258,
         0.98466842, -0.36439522],
       [ 0.69675026,  2.98746598,  0.0673194 , -0.54048391, -0.94477258,
         3.15643938,  0.40099474],
       [-0.76255865, -0.48014277, -0.35441529, -0.17946936,  1.05845579,
         0.1493719 , -0.30505777],
       [ 0.41633404, -0.48014277, -0.35441529,  4.87473439, -0.94477258,
        -0.51886532, -0.37007646]])
```

Train Test Split

In [135…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (6400, 7) (6400,)
Test set: (1600, 7) (1600,)
```

Modelling

Three ML algorithms will be deployed - Decision Tree and Multi Layer Perceptron - MLP

Defining the **function** for plotting. This will also be used in the next ML algorithms to generate the distribution plots for every model.

Decision Tree

Setting up the Decision Tree

In [163…
```
from sklearn.tree import DecisionTreeClassifier
```

Modeling

Firstly I create an instance of the DecisionTreeClassifier called drugTree. Inside of the classifier, I specify criterion="entropy" so the information gain of each node is visible.

In [164…
```
depositTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
depositTree # it shows the default parameters
```

Out[164]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,

                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Next, I fit the data with the training feature matrix X_train and training response vector y_train

In [165…
```
depositTree.fit(X_train,y_train)
```

Out[165]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,

                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Prediction

Predictions on the testing dataset, this is stored into a variable called predTree.

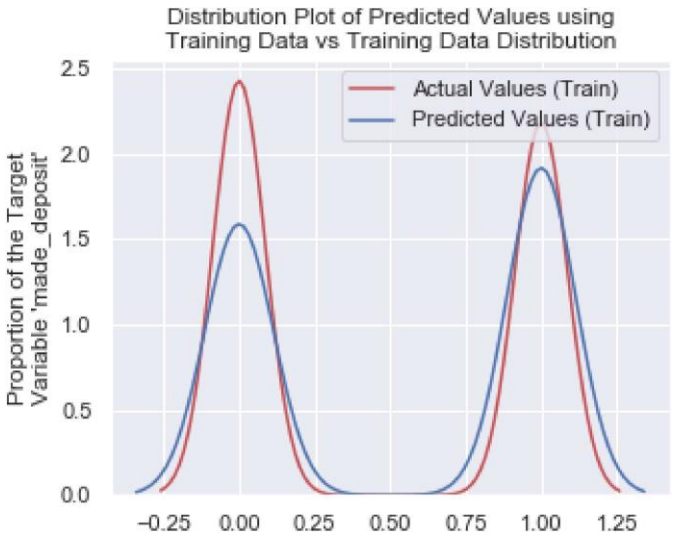
In [166…
```
predTree = depositTree.predict(X_test)
```

In [167…
```
print (predTree [0:5])

print (y_test [0:5])
```

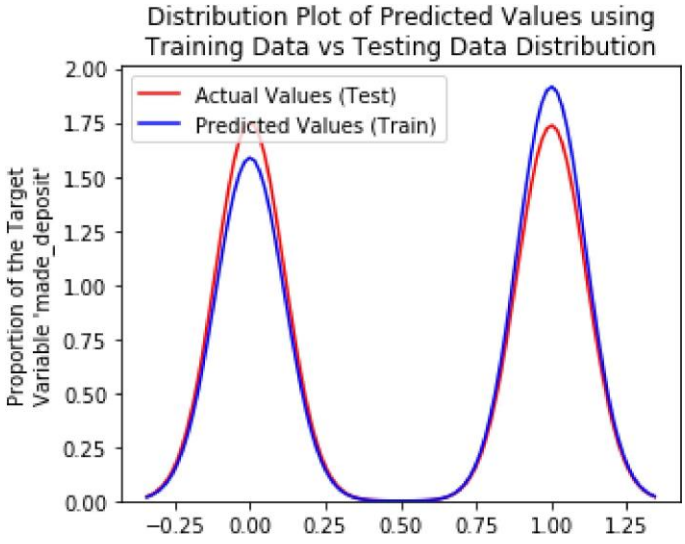
```
[1 1 0 0 1]
[1 1 0 1 1]
```

Distributions of predictions

In [168…
```
Title = 'Distribution Plot of Predicted Values using\nTraining Data vs Training Data D
DistributionPlot(y_train, predTree, "Actual Values (Train)", "Predicted Values (Train)
```



In [344…
```
Title = 'Distribution Plot of Predicted Values using\nTraining Data vs Testing Data Di
DistributionPlot(y_test, predTree, "Actual Values (Test)", "Predicted Values (Train)",
```



Evaluation

Next, importing metrics from sklearn and checking the accuracy of the model.

In [170…
```
from sklearn import metrics

print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, predTree))
```

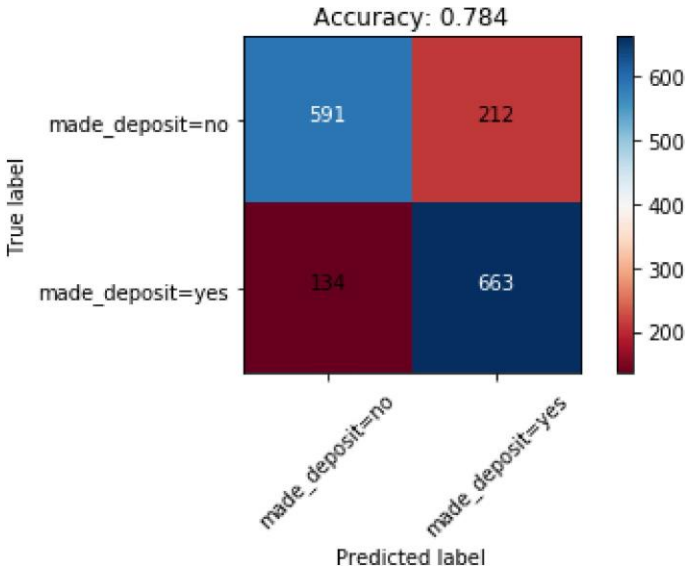
```
DecisionTrees's Accuracy:  0.78375
```

Decision Tree: accuracy of the built model using various evaluation metrics. **Alternatives for evaluation** as previously have been used (f1_score and jaccard index).

Confusion Matrix (Decision Tree)

In [289…
```
# Compute confusion matrix
cnf_matrix_tree = confusion_matrix(y_test, predTree, labels=[0,1])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_tree, classes=['made_deposit=no','made_deposit=yes'],
```

```
Confusion matrix, without normalization
[[591 212]
 [134 663]]
```



Visualization of the decision tree

In [104…
```
# Install the pydotplus and graphviz libraries if have not been installed before
!conda install -c conda-forge pydotplus -y
!conda install -c conda-forge python-graphviz -y
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /Users/stella/miniconda3

  added / updated specs:
    - pydotplus


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2020.4.5.1 |       hecc5488_0         146 KB  conda-forge
    cairo-1.16.0               |    h0ab9d94_1001         1.3 MB  conda-forge
    certifi-2020.4.5.1         |   py37hc8dfbb8_0         151 KB  conda-forge
    conda-4.8.3                |   py37hc8dfbb8_1         3.0 MB  conda-forge
    fontconfig-2.13.1          |    h1027ab8_1000         269 KB  conda-forge
    fribidi-1.0.9              |       h0b31af3_0          64 KB  conda-forge
    graphite2-1.3.13           |    h2098e52_1000          84 KB  conda-forge
    graphviz-2.40.1            |       hefbbd9a_2         6.3 MB
    harfbuzz-1.8.8             |       hb8d4a28_0         414 KB
    libtiff-4.0.9              |       he6b73bb_1         556 KB  conda-forge
    libxml2-2.9.9              |       hf6e021a_1         1.3 MB
    openssl-1.1.1f             |       h0b31af3_0         1.9 MB  conda-forge
    pango-1.42.4               |       h060686c_0         455 KB
    pixman-0.38.0              |    h01d97ff_1003         611 KB  conda-forge
    pydotplus-2.0.2            |     pyhd1c1de3_3          23 KB  conda-forge
    python_abi-3.7             |          1_cp37m           4 KB  conda-forge
    ------------------------------------------------------------
                                           Total:        16.6 MB

The following NEW packages will be INSTALLED:

  cairo              conda-forge/osx-64::cairo-1.16.0-h0ab9d94_1001
  fontconfig         conda-forge/osx-64::fontconfig-2.13.1-h1027ab8_1000
  fribidi            conda-forge/osx-64::fribidi-1.0.9-h0b31af3_0
  graphite2          conda-forge/osx-64::graphite2-1.3.13-h2098e52_1000
  graphviz           pkgs/main/osx-64::graphviz-2.40.1-hefbbd9a_2
  harfbuzz           pkgs/main/osx-64::harfbuzz-1.8.8-hb8d4a28_0
  libtiff            conda-forge/osx-64::libtiff-4.0.9-he6b73bb_1
  libxml2            pkgs/main/osx-64::libxml2-2.9.9-hf6e021a_1
  pango              pkgs/main/osx-64::pango-1.42.4-h060686c_0
  pixman             conda-forge/osx-64::pixman-0.38.0-h01d97ff_1003
  pydotplus          conda-forge/noarch::pydotplus-2.0.2-pyhd1c1de3_3
  python_abi         conda-forge/osx-64::python_abi-3.7-1_cp37m

The following packages will be UPDATED:

  ca-certificates      anaconda::ca-certificates-2020.1.1-0 --> conda-forge::ca-certi
ficates-2020.4.5.1-hecc5488_0
  certifi               anaconda::certifi-2019.11.28-py37_1 --> conda-forge::certifi-
2020.4.5.1-py37hc8dfbb8_0
  conda                       anaconda::conda-4.8.3-py37_0 --> conda-forge::conda-4.
8.3-py37hc8dfbb8_1
```

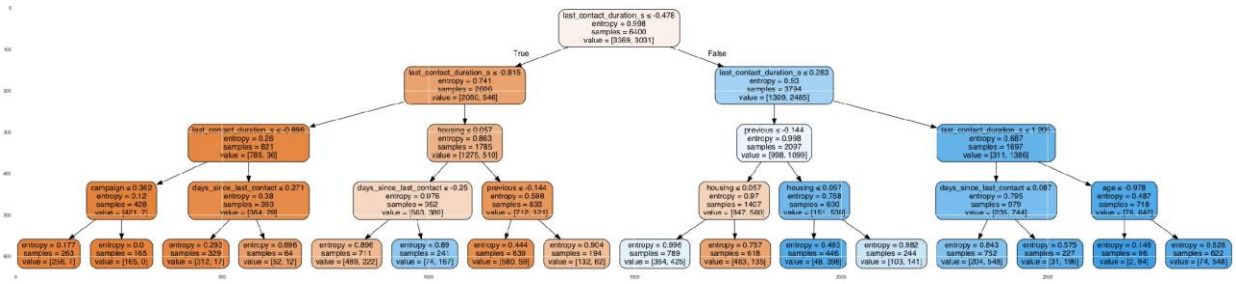
```
The following packages will be SUPERSEDED by a higher-priority channel:

  openssl                anaconda::openssl-1.1.1-h1de35cc_0 --> conda-forge::openssl-
1.1.1f-h0b31af3_0



Downloading and Extracting Packages
libtiff-4.0.9        | 556 KB    | ##################################### | 100%
graphite2-1.3.13     | 84 KB     | ##################################### | 100%
cairo-1.16.0         | 1.3 MB    | ##################################### | 100%
pixman-0.38.0        | 611 KB    | ##################################### | 100%
ca-certificates-2020 | 146 KB    | ##################################### | 100%
pango-1.42.4         | 455 KB    | ##################################### | 100%
fribidi-1.0.9        | 64 KB     | ##################################### | 100%
fontconfig-2.13.1    | 269 KB    | ##################################### | 100%
pydotplus-2.0.2      | 23 KB     | ##################################### | 100%
python_abi-3.7       | 4 KB      | ##################################### | 100%
certifi-2020.4.5.1   | 151 KB    | ##################################### | 100%
openssl-1.1.1f       | 1.9 MB    | ##################################### | 100%
conda-4.8.3          | 3.0 MB    | ##################################### | 100%
graphviz-2.40.1      | 6.3 MB    | ##################################### | 100%
libxml2-2.9.9        | 1.3 MB    | ##################################### | 100%
harfbuzz-1.8.8       | 414 KB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /Users/stella/miniconda3

  added / updated specs:
    - python-graphviz


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    python-graphviz-0.13.2     |             py_0          18 KB  conda-forge
                                           Total:          18 KB

The following NEW packages will be INSTALLED:

  python-graphviz    conda-forge/noarch::python-graphviz-0.13.2-py_0



Downloading and Extracting Packages
python-graphviz-0.13 | 18 KB     | ##################################### | 100%
Preparing transaction: done
```

```
Verifying transaction: done
Executing transaction: done
```

In [176…
```
# Importing libraries for visualizing the tree

from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [178…
```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier


features = ['{}'.format(c) for c in Feature]

# viz
%matplotlib inline
import sklearn
from sklearn.tree import export_graphviz

import seaborn as sns

dot_data = StringIO()
export_graphviz(depositTree, out_file=dot_data,
                    feature_names=features,
#                     class_names=tree.classes_,
                     filled=True, rounded=True,
                    special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(50, 100))
plt.imshow(img,interpolation='nearest')
```

Out[178]:
```
<matplotlib.image.AxesImage at 0x1a23ea0a50>
```



Multi Layer Perceptron - MLP

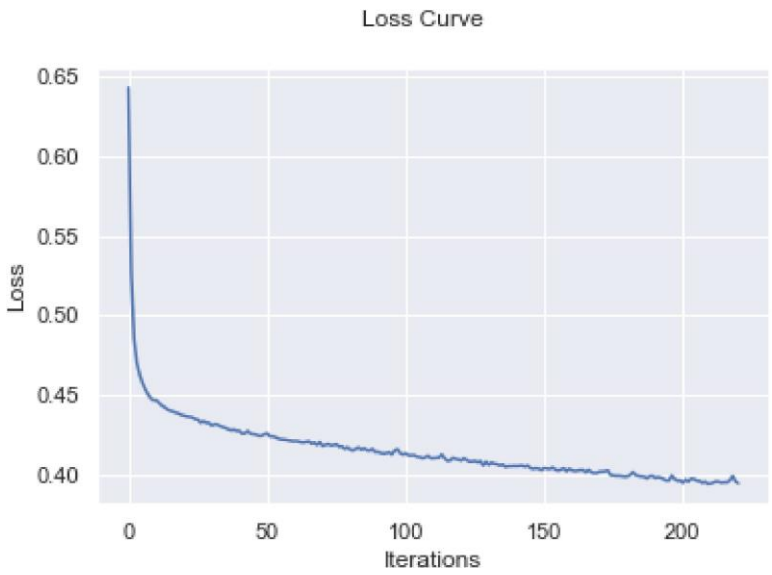
In [181…
```
#train_test_split has been performed previously
```

In [199…
```
mlp = MLPClassifier(hidden_layer_sizes=(32, 100), max_iter=500, alpha=0.0001, activati
                    solver='adam', verbose='True', shuffle=True, random_state=21, tol
```

The loss curve

In [202…
```
loss_values = mlp.loss_curve_

fig = plt.figure()
plt.plot(loss_values)
fig.suptitle('Loss Curve')
plt.xlabel('Iterations')
plt.ylabel('Loss')
```

Out[202]:
```
Text(0, 0.5, 'Loss')
```

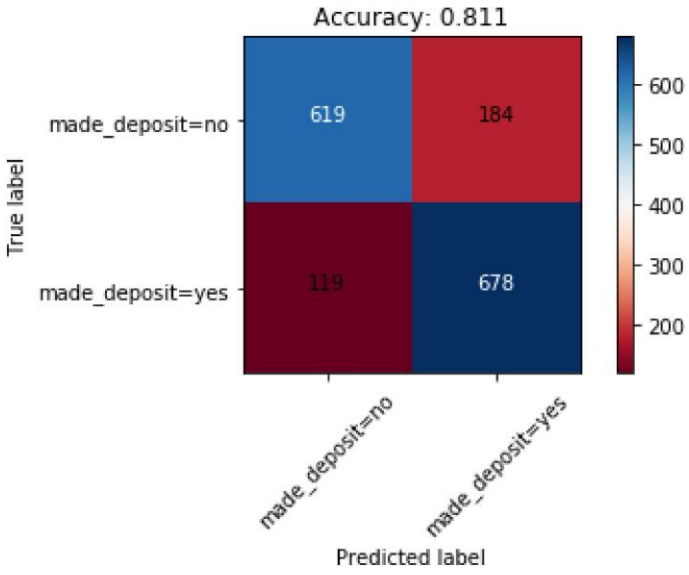


In [322…
```
# Compute confusion matrix
cnf_matrix_mlp = confusion_matrix(y_test, y_pred, labels=[0,1])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix_mlp, classes=['made_deposit=no','made_deposit=yes'],r
```

```
Confusion matrix, without normalization
```

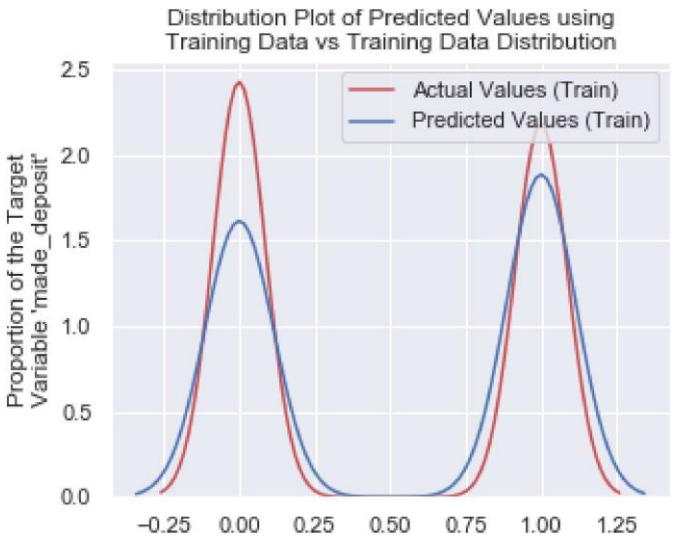
In [180…
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
[[619 184]
 [119 678]]
```

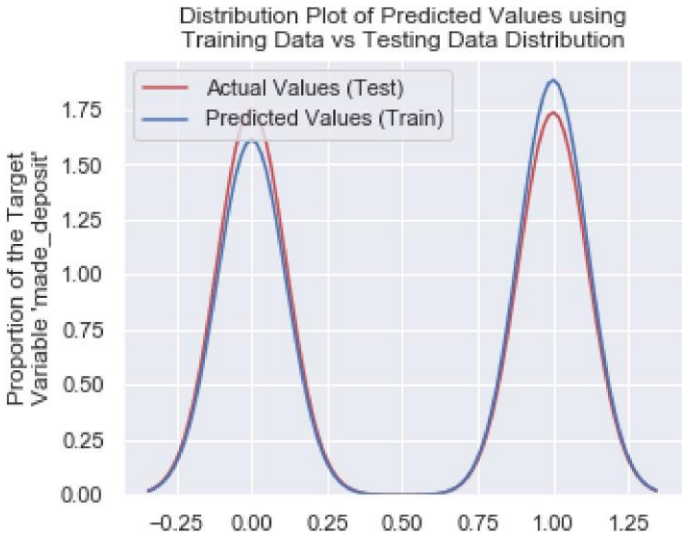


Distribution Plots

In [188…
```
Title = 'Distribution Plot of Predicted Values using\nTraining Data vs Training Data D
DistributionPlot(y_train, y_pred, "Actual Values (Train)", "Predicted Values (Train)",
```



In [189…
```
Title = 'Distribution Plot of Predicted Values using\nTraining Data vs Testing Data Di
DistributionPlot(y_test, y_pred, "Actual Values (Test)", "Predicted Values (Train)", T
```



ROC Curve

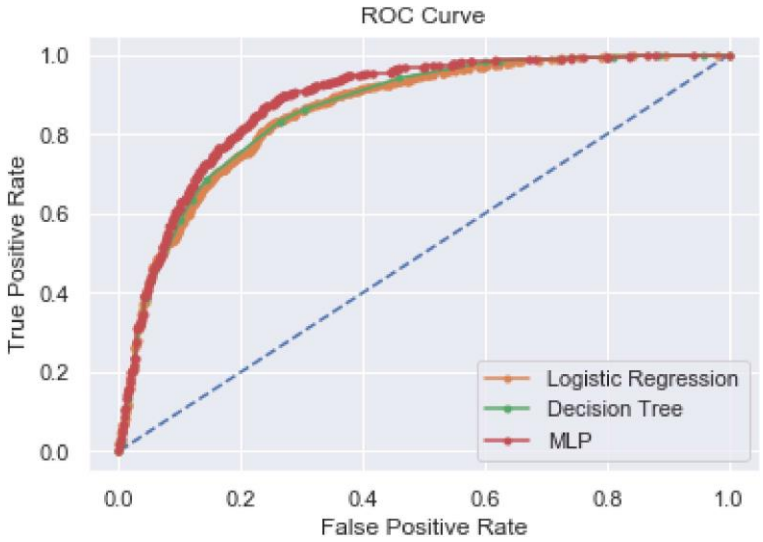
In [190…

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

def display_roc(models, x, y):

    ns_probs = [0 for _ in range(len(y))]
    ns_fpr, ns_tpr, _ = roc_curve(y, ns_probs)
    plt.plot(ns_fpr, ns_tpr, linestyle='--', label='')

    for name, model in models.items():

        lr_probs = model.predict_proba(x)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y, ns_probs)
        lr_auc = roc_auc_score(y, lr_probs)

        print('%s: ROC AUC=%.3f' % (name, lr_auc))
        lr_fpr, lr_tpr, _ = roc_curve(y, lr_probs)
        plt.plot(lr_fpr, lr_tpr, marker='.', label=name)

    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')

    plt.legend()
    plt.show()
```

In [203…

```
models = {
    'Logistic Regression': LR,
    'Decision Tree': depositTree,
    'MLP': mlp
}

display_roc(models, X_test, y_test)
```

```
     Logistic Regression: ROC AUC=0.860
Decision Tree: ROC AUC=0.861
     MLP: ROC AUC=0.882
```



Report

Reporting the accuracy of the built models using different evaluation metrics:(Logistic Regression technique values are taken in general)

| Algorithm | Jaccard | F1-score | LogLoss | |--------------------|--------- |---------- |--------- | | Logistic Regression|0.77125 |0.77037 | 0.492416 | | Decision Tree |0.78375 |0.78327 | NA | | MLP |0.81062 |0.81034 | NA |