Experiment 7 - Bank Marketing Data - A Decision Tree Approach

The aim of this attempt is to predict if the client will subscribe (yes/no) to a term deposit, by building a classification model using Decision Tree. ___

Import Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn import datasets
from io import StringIO
from sklearn.tree import export_graphviz
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics
%matplotlib inline
```

```
/home/volt/.local/lib/python3.10/site-packages/scipy/__init__.py:146: UserW
arning: A NumPy version >=1.16.5 and <1.23.0 is required for this version o
f SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Read csv file

In [ ]:
```
# Load data file
bank=pd.read_csv('./data/decision-tree-bank.csv')
bank.head()
```

Out[ ]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	d
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	

Summary of data:

1. Categorical Variables :

- job : admin,technician, services, management, retired, blue-collar, unemployed, entrepreneur, housemaid, unknown, self-employed, student
- marital : married, single, divorced
- education: secondary, tertiary, primary, unknown
- default : yes, no
- housing : yes, no
- loan : yes, no
- deposit : yes, no (Dependent Variable)
- contact : unknown, cellular, telephone
- month : jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
- poutcome: unknown, other, failure, success

2. Numerical Variables:

- age
- balance
- day
- duration
- campaign
- pdays
- previous

In [ ]:
```
# Check if the data set contains any null values - Nothing found!
bank[bank.isnull().any(axis=1)].count()
```

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

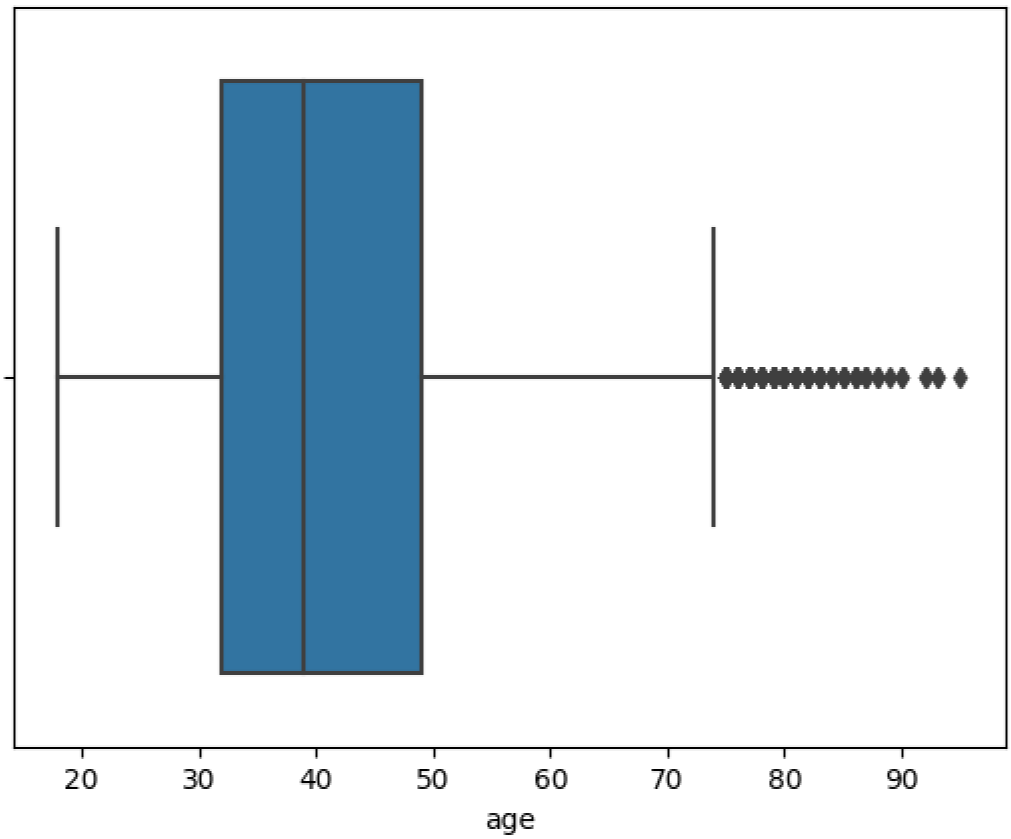
In [ ]:
```
bank.describe()
```

Out[ ]:

	age	balance	day	duration	campaign	pdays	
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	1116
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330407	
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758282	
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000	
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000	
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000	
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000	
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000	5

In [ ]:

```
# Boxplot for 'age'
g = sns.boxplot(x=bank["age"])
```

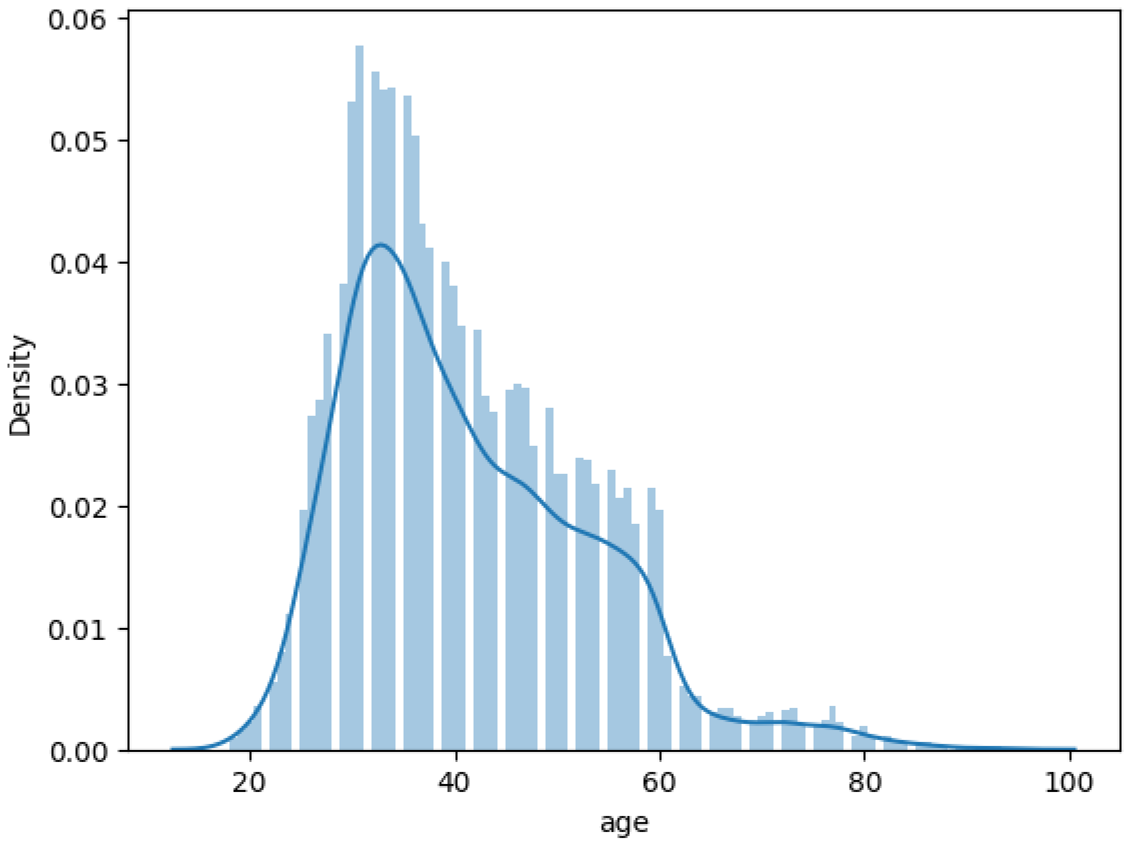


In [ ]:

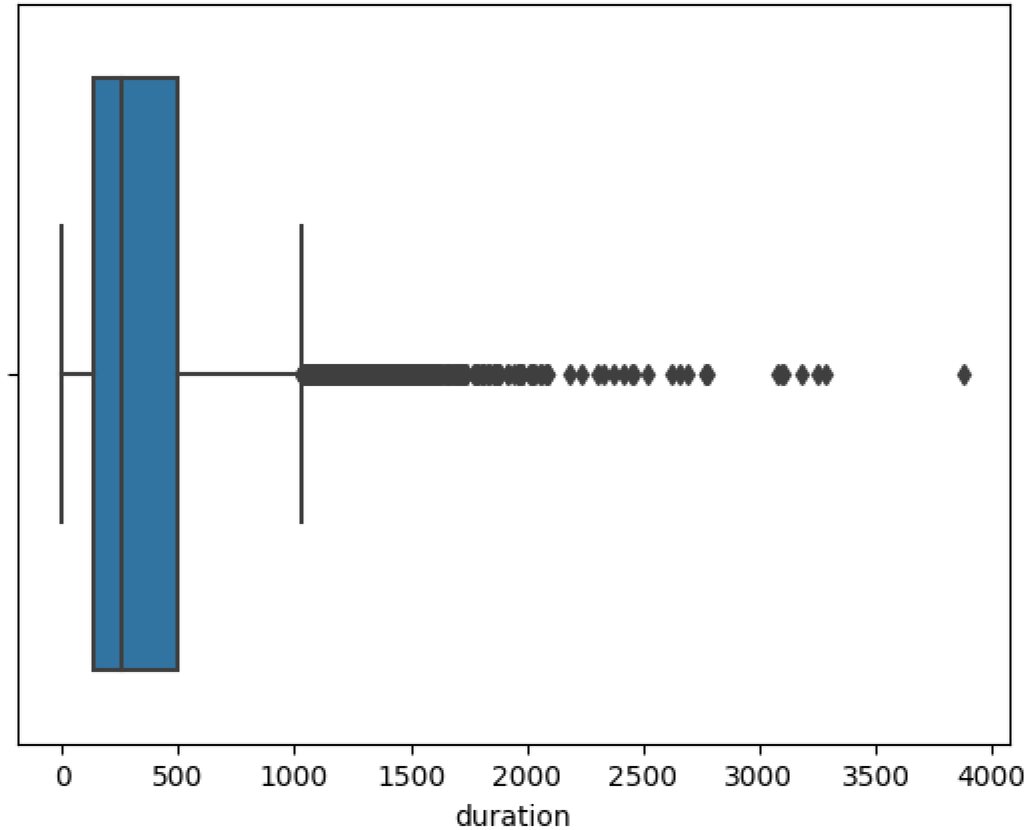
```
# Distribution of Age
sns.distplot(bank.age, bins=100)
```

```
/tmp/ipykernel_34465/2873891249.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function wit
h
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(bank.age, bins=100)
```

Out[ ]: <AxesSubplot: xlabel='age', ylabel='Density'>



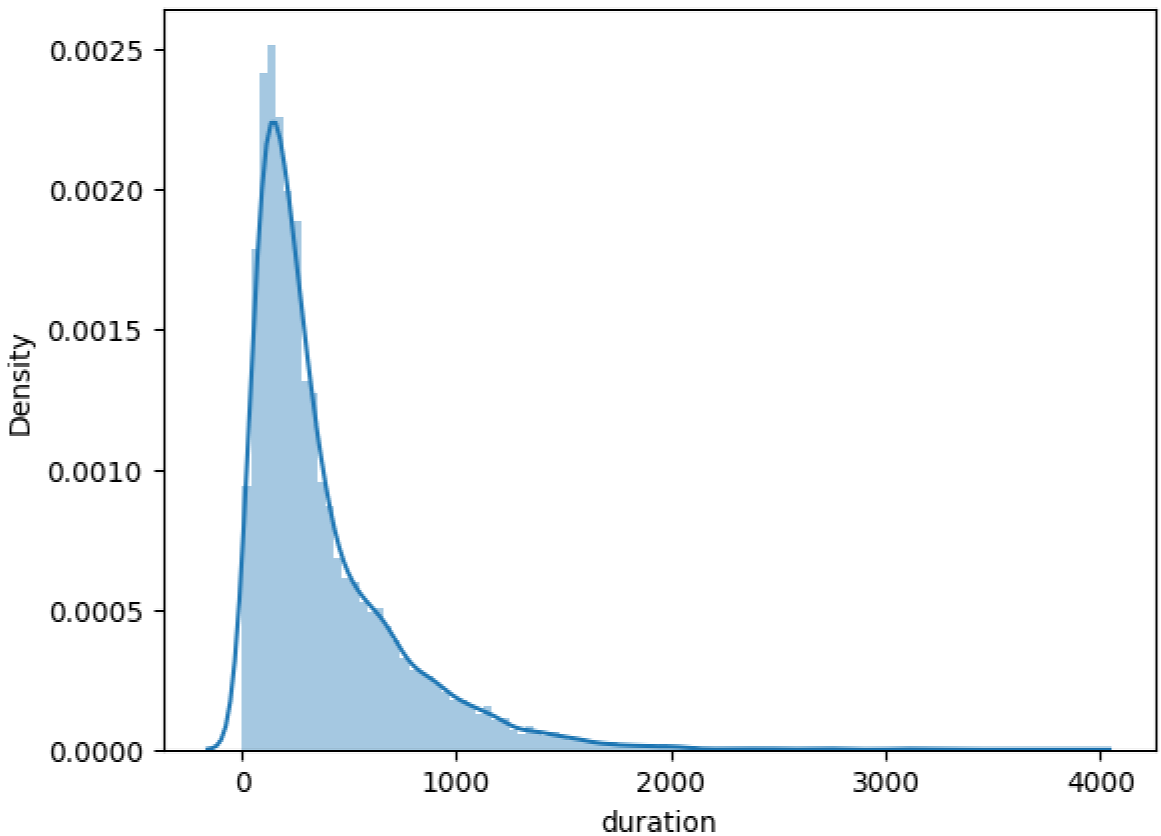
In [ ]:
```
# Boxplot for 'duration'
g = sns.boxplot(x=bank["duration"])
```



In [ ]:
```
sns.distplot(bank.duration, bins=100)
```

```
/tmp/ipykernel_34465/1784427431.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function wit
h
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(bank.duration, bins=100)
```

Out[ ]: <AxesSubplot: xlabel='duration', ylabel='Density'>



Convert categorical data

In [ ]:
```
# Make a copy for parsing
bank_data = bank.copy()
```

Job

In [ ]:
```
# Explore People who made a deposit Vs Job category
jobs = ['management','blue-collar','technician','admin.','services','retirec
        'unemployed','entrepreneur','housemaid','unknown']

for j in jobs:
    print("{:15} : {:5}". format(j, len(bank_data[(bank_data.deposit == "yes
```

```
management      :  1301
blue-collar     :   708
technician      :   840
admin.          :   631
services        :   369
retired         :   516
self-employed   :   187
student         :   269
unemployed      :   202
entrepreneur    :   123
housemaid       :   109
unknown         :    34
```

In [ ]:
```
# Different types of job categories and their counts
bank_data.job.value_counts()
```

Out[ ]:
```
management       2566
blue-collar      1944
technician       1823
admin.           1334
services          923
retired           778
self-employed     405
student           360
unemployed        357
entrepreneur      328
housemaid         274
unknown            70
Name: job, dtype: int64
```

In [ ]:
```
# Combine similar jobs into categiroes
bank_data['job'] = bank_data['job'].replace(['management', 'admin.'], 'white
bank_data['job'] = bank_data['job'].replace(['services','housemaid'], 'pink-
bank_data['job'] = bank_data['job'].replace(['retired', 'student', 'unemploy
```

In [ ]:
```
# New value counts
bank_data.job.value_counts()
```

Out[ ]:
```
white-collar     3900
blue-collar      1944
technician       1823
other            1565
pink-collar      1197
self-employed     405
entrepreneur      328
Name: job, dtype: int64
```

poutcome

In [ ]:
```
bank_data.poutcome.value_counts()
```

Out[ ]:
```
unknown    8326
failure    1228
success    1071
other       537
Name: poutcome, dtype: int64
```

In [ ]:
```
# Combine 'unknown' and 'other' as 'other' isn't really match with either 's
bank_data['poutcome'] = bank_data['poutcome'].replace(['other'] , 'unknown')
bank_data.poutcome.value_counts()
```

Out[ ]:
```
unknown    8863
failure    1228
success    1071
Name: poutcome, dtype: int64
```

contact

In [ ]:
```
# Drop 'contact', as every participant has been contacted.
bank_data.drop('contact', axis=1, inplace=True)
```

default

In [ ]:
```
# values for "default" : yes/no
bank_data["default"]
bank_data['default_cat'] = bank_data['default'].map( {'yes':1, 'no':0} )
bank_data.drop('default', axis=1,inplace = True)
```

housing

In [ ]:
```
# values for "housing" : yes/no
bank_data["housing_cat"]=bank_data['housing'].map({'yes':1, 'no':0})
bank_data.drop('housing', axis=1,inplace = True)
```

loan

In [ ]:
```
# values for "loan" : yes/no
bank_data["loan_cat"] = bank_data['loan'].map({'yes':1, 'no':0})
bank_data.drop('loan', axis=1, inplace=True)
```

month, day

In [ ]:
```
# day  : last contact day of the month
# month: last contact month of year
# Drop 'month' and 'day' as they don't have any intrinsic meaning
bank_data.drop('month', axis=1, inplace=True)
bank_data.drop('day', axis=1, inplace=True)
```

deposit

In [ ]:
```
# values for "deposit" : yes/no
bank_data["deposit_cat"] = bank_data['deposit'].map({'yes':1, 'no':0})
bank_data.drop('deposit', axis=1, inplace=True)
```

pdays

In [ ]:
```
# pdays: number of days that passed by after the client was last contacted f
#       -1 means client was not previously contacted

print("Customers that have not been contacted before:", len(bank_data[bank_c
print("Maximum values on padys    :", bank_data['pdays'].max())
```

```
Customers that have not been contacted before: 8324
Maximum values on padys    : 854
```

In [ ]:
```
# Map padys=-1 into a large value (10000 is used) to indicate that it is so
bank_data.loc[bank_data['pdays'] == -1, 'pdays'] = 10000
```

In [ ]:
```
# Create a new column: recent_pdays
bank_data['recent_pdays'] = np.where(bank_data['pdays'], 1/bank_data.pdays,

# Drop 'pdays'
bank_data.drop('pdays', axis=1, inplace = True)
```

In [ ]:
```
bank_data.tail()
```

Out[ ]:

	age	job	marital	education	balance	duration	campaign	previous	poutcome	de
11157	33	blue-collar	single	primary	1	257	1	0	unknown	
11158	39	pink-collar	married	secondary	733	83	4	0	unknown	
11159	32	technician	single	secondary	29	156	2	0	unknown	
11160	43	technician	married	secondary	0	9	2	5	failure	
11161	34	technician	married	secondary	0	628	1	0	unknown	

convert to dummy variables

In [ ]:
```
# Convert categorical variables to dummies
bank_with_dummies = pd.get_dummies(data=bank_data, columns = ['job', 'marita
                                   prefix = ['job', 'marital', 'education',
bank_with_dummies.head()
```

Out[ ]:

	age	balance	duration	campaign	previous	default_cat	housing_cat	loan_cat	deposit_cat
0	59	2343	1042	1	0	0	1	0	1
1	56	45	1467	1	0	0	0	0	1
2	41	1270	1389	1	0	0	1	0	1
3	55	2476	579	1	0	0	1	0	1
4	54	184	673	2	0	0	0	0	1

5 rows × 27 columns

In [ ]:
```
bank_with_dummies.shape
```

Out[ ]: (11162, 27)

In [ ]:
```
bank_with_dummies.describe()
```

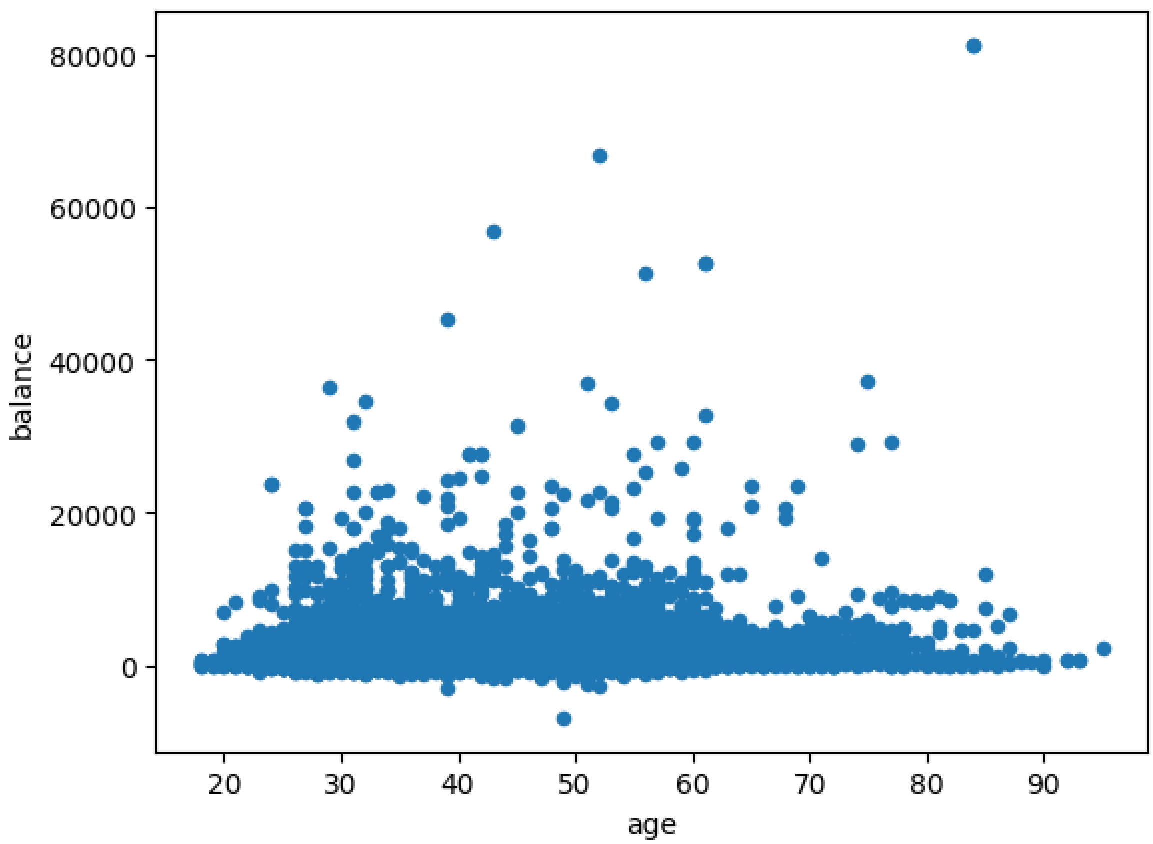
Out[ ]:

	age	balance	duration	campaign	previous	default_cat	hou
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	1116
mean	41.231948	1528.538524	371.993818	2.508421	0.832557	0.015051	
std	11.913369	3225.413326	347.128386	2.722077	2.292007	0.121761	
min	18.000000	-6847.000000	2.000000	1.000000	0.000000	0.000000	
25%	32.000000	122.000000	138.000000	1.000000	0.000000	0.000000	
50%	39.000000	550.000000	255.000000	2.000000	0.000000	0.000000	
75%	49.000000	1708.000000	496.000000	3.000000	1.000000	0.000000	
max	95.000000	81204.000000	3881.000000	63.000000	58.000000	1.000000	

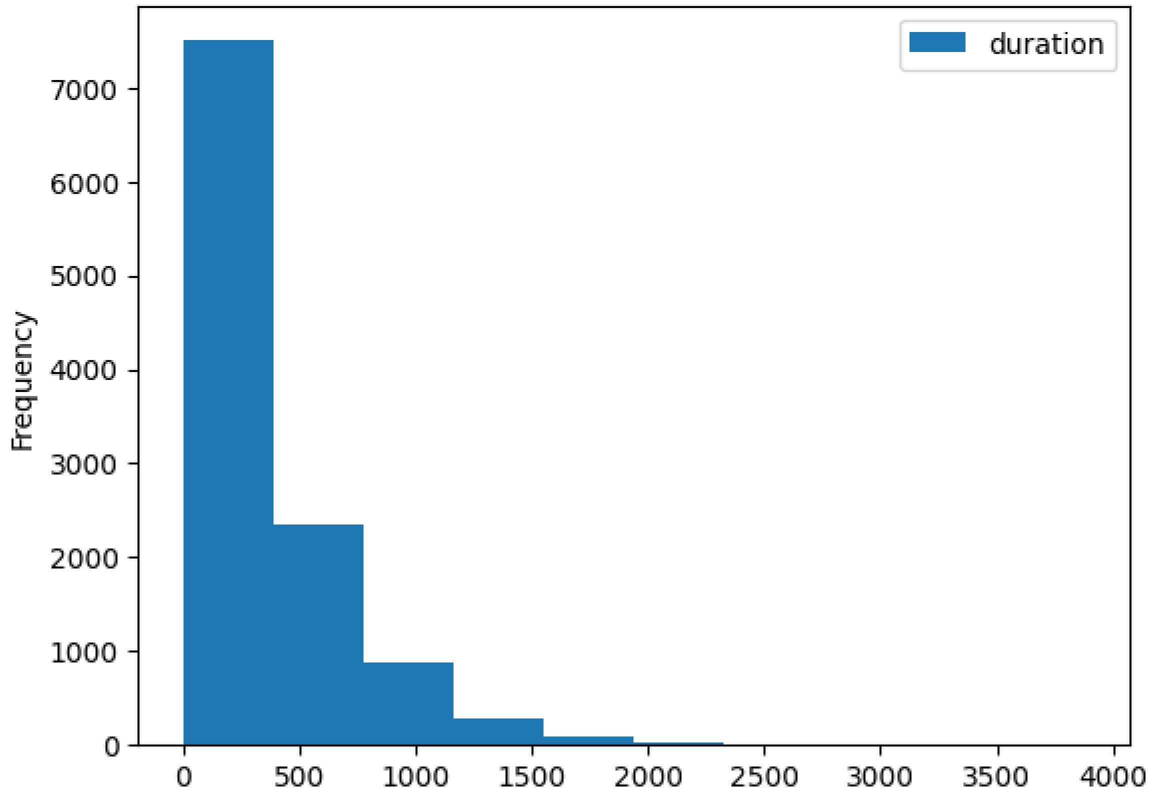
8 rows × 27 columns

Observations on whole population

In [ ]:
```
# Scatter-plot showing age and balance
bank_with_dummies.plot(kind='scatter', x='age', y='balance');

# Across all ages, majority of people have savings of less than 20000.
```



In [ ]:
```
bank_with_dummies.plot(kind='hist', x='poutcome_success', y='duration');
```



Analysis on people who sign up for a term deposit

In [ ]:
```
# People who sign up to a term deposite
bank_with_dummies[bank_data.deposit_cat == 1].describe()
```

Out[ ]:

	age	balance	duration	campaign	previous	default_cat	housing_
count	5289.000000	5289.000000	5289.000000	5289.000000	5289.000000	5289.000000	5289.00(
mean	41.670070	1804.267915	537.294574	2.141047	1.170354	0.009832	0.365
std	13.497781	3501.104777	392.525262	1.921826	2.553272	0.098676	0.481
min	18.000000	-3058.000000	8.000000	1.000000	0.000000	0.000000	0.00(
25%	31.000000	210.000000	244.000000	1.000000	0.000000	0.000000	0.00(
50%	38.000000	733.000000	426.000000	2.000000	0.000000	0.000000	0.00(
75%	50.000000	2159.000000	725.000000	3.000000	1.000000	0.000000	1.00(
max	95.000000	81204.000000	3881.000000	32.000000	58.000000	1.000000	1.00(

8 rows × 27 columns

In [ ]:
```
# People signed up to a term deposit having a personal loan (loan_cat) and h
len(bank_with_dummies[(bank_with_dummies.deposit_cat == 1) & (bank_with_dumm
```

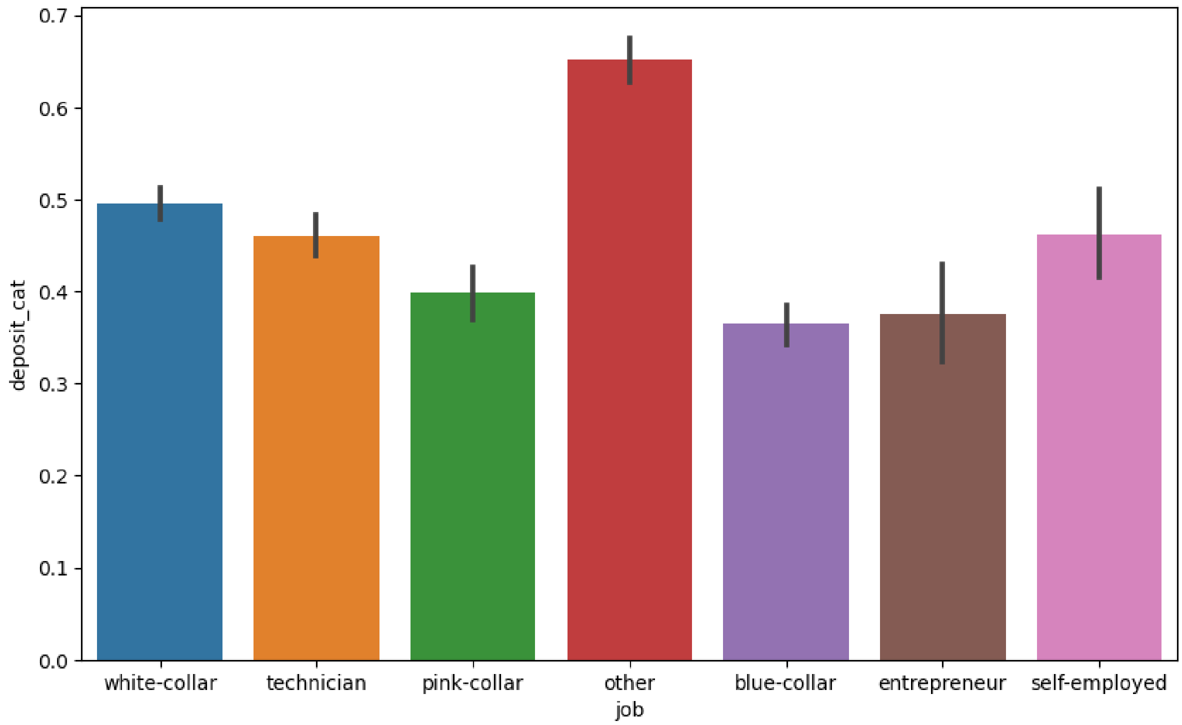
Out[ ]: 265

In [ ]:
```
# People signed up to a term deposit with a credit default
len(bank_with_dummies[(bank_with_dummies.deposit_cat == 1) & (bank_with_dumm
```

Out[ ]: 52

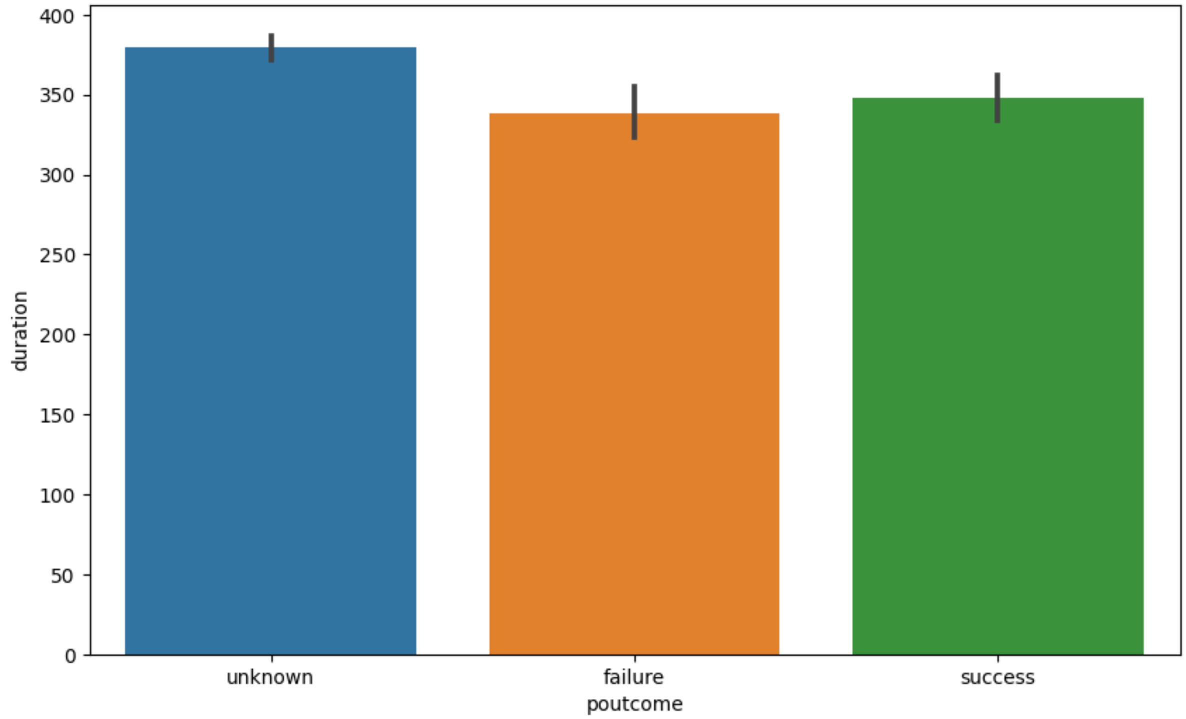
In [ ]:
```
# Bar chart of job Vs deposit
plt.figure(figsize = (10,6))
sns.barplot(x='job', y = 'deposit_cat', data = bank_data)
```

Out[ ]: <AxesSubplot: xlabel='job', ylabel='deposit_cat'>



In [ ]:
```
# Bar chart of "previous outcome" Vs "call duration"

plt.figure(figsize = (10,6))
sns.barplot(x='poutcome', y = 'duration', data = bank_data)
```

Out[ ]: <AxesSubplot: xlabel='poutcome', ylabel='duration'>



Classification

In [ ]:
```
# make a copy
bankcl = bank_with_dummies
```

In [ ]:
```
# The Correlation matrix
corr = bankcl.corr()
corr
```

Out[ ]:

	age	balance	duration	campaign	previous	default_cat	housing_
age	1.000000	0.112300	0.000189	-0.005278	0.020169	-0.011425	-0.168
balance	0.112300	1.000000	0.022436	-0.013894	0.030805	-0.060954	-0.077
duration	0.000189	0.022436	1.000000	-0.041557	-0.026716	-0.009760	0.035
campaign	-0.005278	-0.013894	-0.041557	1.000000	-0.049699	0.030975	0.006
previous	0.020169	0.030805	-0.026716	-0.049699	1.000000	-0.035273	-0.000
default_cat	-0.011425	-0.060954	-0.009760	0.030975	-0.035273	1.000000	0.011
housing_cat	-0.168700	-0.077092	0.035051	0.006660	-0.000840	0.011076	1.000
loan_cat	-0.031418	-0.084589	-0.001914	0.034722	-0.022668	0.076434	0.076
deposit_cat	0.034901	0.081129	0.451919	-0.128081	0.139867	-0.040680	-0.203
recent_pdays	0.019102	-0.004379	-0.014868	-0.026296	0.122076	-0.011290	-0.029
job_blue-collar	-0.066567	-0.046220	0.029986	0.005522	-0.039939	0.022779	0.189
job_entrepreneur	0.024176	0.005039	-0.000908	0.013883	-0.022470	0.022060	0.011
job_other	0.296418	0.050744	0.010680	-0.050212	0.031191	-0.018130	-0.233
job_pink-collar	-0.027942	-0.041063	0.005345	0.011958	-0.028623	-0.007173	0.043
job_self-employed	-0.023163	0.020264	0.013506	0.001776	-0.002338	0.007493	-0.016
job_technician	-0.082716	0.003802	-0.010440	0.021738	0.002035	0.003109	0.006
job_white-collar	-0.080122	0.013780	-0.031980	0.001944	0.034929	-0.013425	-0.012
marital_divorced	0.186349	-0.017586	0.021364	-0.006828	-0.026566	0.019633	0.007
marital_married	0.318436	0.025431	-0.036179	0.047722	-0.005176	-0.006819	0.036
marital_single	-0.467799	-0.014994	0.023847	-0.046165	0.023817	-0.006255	-0.043
education_primary	0.231150	-0.000673	0.013405	0.019915	-0.024852	0.013858	0.017
education_secondary	-0.094400	-0.070609	0.003820	-0.013834	-0.004620	-0.000618	0.118
education_tertiary	-0.101372	0.069128	-0.006813	-0.005427	0.028146	-0.011768	-0.114
education_unknown	0.077761	0.014596	-0.015887	0.012976	-0.011898	0.005421	-0.053
poutcome_failure	-0.008071	0.001695	-0.033966	-0.080188	0.335870	-0.024650	0.087
poutcome_success	0.062114	0.045603	-0.022578	-0.091807	0.325477	-0.040272	-0.136
poutcome_unknown	-0.038992	-0.034524	0.042725	0.128907	-0.496921	0.048403	0.031

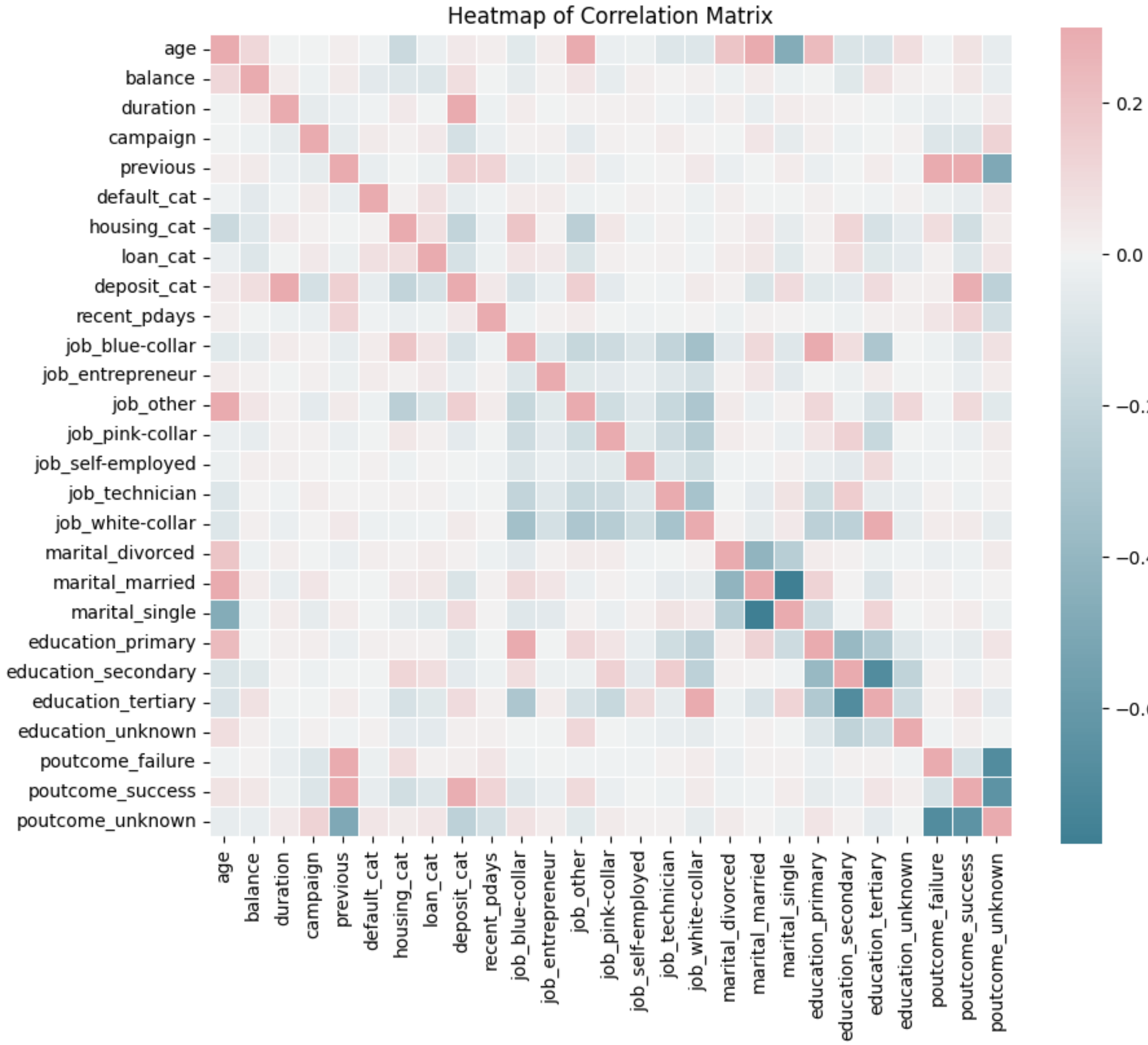
27 rows × 27 columns

In [ ]:

```
# Heatmap
plt.figure(figsize = (10,10))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.
plt.title('Heatmap of Correlation Matrix')
```

Out[ ]:

```
Text(0.5, 1.0, 'Heatmap of Correlation Matrix')
```



In [ ]:

```
# Extract the deposte_cat column (the dependent variable)
corr_deposite = pd.DataFrame(corr['deposit_cat'].drop('deposit_cat'))
corr_deposite.sort_values(by = 'deposit_cat', ascending = False)
```

Out[ ]:

	deposit_cat
duration	0.451919
poutcome_success	0.286642
job_other	0.144408
previous	0.139867
marital_single	0.094632
education_tertiary	0.094598
balance	0.081129
age	0.034901
recent_pdays	0.034457
job_white-collar	0.031621
poutcome_failure	0.020714
education_unknown	0.014355
marital_divorced	0.005228
job_self-employed	-0.004707
job_technician	-0.011557
job_entrepreneur	-0.034443
default_cat	-0.040680
job_pink-collar	-0.051717
education_secondary	-0.051952
education_primary	-0.063002
marital_married	-0.092157
job_blue-collar	-0.100840
loan_cat	-0.110580
campaign	-0.128081
housing_cat	-0.203888
poutcome_unknown	-0.224785

Build the Data Model

In [ ]:
```
# Train-Test split: 20% test data
data_drop_deposite = bankcl.drop('deposit_cat', axis=1)
label = bankcl.deposit_cat
data_train, data_test, label_train, label_test = train_test_split(data_drop_
```

In [ ]:
```
# Decision tree with depth = 2
dt2 = tree.DecisionTreeClassifier(random_state=1, max_depth=2)
dt2.fit(data_train, label_train)
dt2_score_train = dt2.score(data_train, label_train)
print("Training score: ",dt2_score_train)
```

```
dt2_score_test = dt2.score(data_test, label_test)
print("Testing score: ",dt2_score_test)
```

```
Training score:  0.7285250307985217
Testing score:  0.7268248992386923
```

In [ ]:
```
# Decision tree with depth = 3
dt3 = tree.DecisionTreeClassifier(random_state=1, max_depth=3)
dt3.fit(data_train, label_train)
dt3_score_train = dt3.score(data_train, label_train)
print("Training score: ",dt3_score_train)
dt3_score_test = dt3.score(data_test, label_test)
print("Testing score: ",dt3_score_test)
```

```
Training score:  0.770411020271027
Testing score:  0.7572772055530677
```

In [ ]:
```
# Decision tree with depth = 4
dt4 = tree.DecisionTreeClassifier(random_state=1, max_depth=4)
dt4.fit(data_train, label_train)
dt4_score_train = dt4.score(data_train, label_train)
print("Training score: ",dt4_score_train)
dt4_score_test = dt4.score(data_test, label_test)
print("Testing score: ",dt4_score_test)
```

```
Training score:  0.7885541494008288
Testing score:  0.774294670846395
```

In [ ]:
```
# Decision tree with depth = 6
dt6 = tree.DecisionTreeClassifier(random_state=1, max_depth=6)
dt6.fit(data_train, label_train)
dt6_score_train = dt6.score(data_train, label_train)
print("Training score: ",dt6_score_train)
dt6_score_test = dt6.score(data_test, label_test)
print("Testing score: ",dt6_score_test)
```

```
Training score:  0.8080412140217269
Testing score:  0.7796686072548141
```

In [ ]:
```
# Decision tree: To the full depth
dt1 = tree.DecisionTreeClassifier()
dt1.fit(data_train, label_train)
dt1_score_train = dt1.score(data_train, label_train)
print("Training score: ", dt1_score_train)
dt1_score_test = dt1.score(data_test, label_test)
print("Testing score: ", dt1_score_test)
```

```
Training score:  1.0
Testing score:  0.7344379758172862
```

Compare Training and Testing scores for various tree depths used

In [ ]:
```
print('{:10} {:20} {:20}'.format('depth', 'Training score','Testing score'))
print('{:10} {:20} {:20}'.format('-----', '--------------','-------------'))
print('{:1} {:>25} {:>20}'.format(2, dt2_score_train, dt2_score_test))
print('{:1} {:>25} {:>20}'.format(3, dt3_score_train, dt3_score_test))
```

```
print('{:1} {:>25} {:>20}'.format(4, dt4_score_train, dt4_score_test))
print('{:1} {:>25} {:>20}'.format(6, dt6_score_train, dt6_score_test))
print('{:1} {:>23} {:>20}'.format("max", dt1_score_train, dt1_score_test))
```

```
depth      Training score       Testing score
-----      --------------       -------------
2        0.7285250307985217   0.7268248992386923
3         0.770411020271027   0.7572772055530677
4        0.7885541494008288    0.774294670846395
6        0.8080412140217269   0.7796686072548141
max                     1.0   0.7344379758172862
```

It could be seen that, higher the depth, training score increases and matches perfects with the training data set. However higher the depth the tree goes, it overfit to the training data set. So it's no use keep increasing the tree depth. According to above observations, tree with a depth of 2 seems more reasonable as both training and test scores are reasonably high.

In [ ]:
```
# Let's generate the decision tree for depth = 2
# Create a feature vector
features = bankcl.columns.tolist()

# Uncomment below to generate the digraph Tree.
#tree.export_graphviz(dt2, out_file='tree_depth_2.dot', feature_names=featur
```

Contents of "tree_depth_2.dot":

- digraph Tree {
- node [shape=box] ;
- 0 [label="duration <= 206.5\ngini = 0.4986\nsamples = 8929\nvalue = [4700, 4229]"] ;
- 1 [label="poutcome_failure <= 0.5\ngini = 0.3274\nsamples = 3612\nvalue = [2867, 745]"] ;
- 0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
- 2 [label="gini = 0.2733\nsamples = 3380\nvalue = [2828, 552]"] ;
- 1 -> 2 ;
- 3 [label="gini = 0.2797\nsamples = 232\nvalue = [39, 193]"] ;
- 1 -> 3 ;
- 4 [label="duration <= 441.5\ngini = 0.4518\nsamples = 5317\nvalue = [1833, 3484]"] ;
- 0 -> 4 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
- 5 [label="gini = 0.4996\nsamples = 2762\nvalue = [1340, 1422]"] ;
- 4 -> 5 ;
- 6 [label="gini = 0.3114\nsamples = 2555\nvalue = [493, 2062]"] ;
- 4 -> 6 ;
- }

Based on the decision tree results, it could be seen that higher the "duration", bank is able to sign up more people to term deposites.

In [ ]:
```
# Two classes: 0 = not signed up,  1 = signed up
dt2.classes_
```

Out[ ]:
```
array([0, 1])
```

In [ ]:
```
# Create a feature vector
features = data_drop_deposite.columns.tolist()

features
```

Out[ ]:
```
['age',
 'balance',
 'duration',
 'campaign',
 'previous',
 'default_cat',
 'housing_cat',
 'loan_cat',
 'recent_pdays',
 'job_blue-collar',
 'job_entrepreneur',
 'job_other',
 'job_pink-collar',
 'job_self-employed',
 'job_technician',
 'job_white-collar',
 'marital_divorced',
 'marital_married',
 'marital_single',
 'education_primary',
 'education_secondary',
 'education_tertiary',
 'education_unknown',
 'poutcome_failure',
 'poutcome_success',
 'poutcome_unknown']
```

In [ ]:
```
# Investigate most important features with depth =2

dt2 = tree.DecisionTreeClassifier(random_state=1, max_depth=2)

# Fit the decision tree classifier
dt2.fit(data_train, label_train)

fi = dt2.feature_importances_

l = len(features)
for i in range(0,len(features)):
    print('{:.<20} {:3}'.format(features[i],fi[i]))
```

```
age................. 0.0
balance............. 0.0
duration............ 0.849306123902405
campaign............ 0.0
previous............ 0.0
default_cat......... 0.0
housing_cat......... 0.0
loan_cat............ 0.0
recent_pdays........ 0.0
job_blue-collar..... 0.0
job_entrepreneur.... 0.0
job_other........... 0.0
job_pink-collar..... 0.0
job_self-employed... 0.0
job_technician...... 0.0
job_white-collar.... 0.0
marital_divorced.... 0.0
marital_married..... 0.0
marital_single...... 0.0
education_primary... 0.0
education_secondary. 0.0
education_tertiary.. 0.0
education_unknown... 0.0
poutcome_failure.... 0.0
poutcome_success.... 0.15069387609759496
poutcome_unknown.... 0.0
```

Predictions

In [ ]:

```
# According to feature importance results, most importtant feature is the "D
# Let's calculte statistics on Duration
print("Mean duration   : ", data_drop_deposite.duration.mean())
print("Maximun duration: ", data_drop_deposite.duration.max())
print("Minimum duration: ", data_drop_deposite.duration.min())
```

```
Mean duration   :  371.99381831213043
Maximun duration:  3881
Minimum duration:  2
```

In [ ]:

```
# Predict: Successful deposite with a call duration = 371 sec

print(dt2.predict_proba(np.array([0, 0, 371, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0
print(dt2.predict(np.array([0, 0, 371, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# column 0: probability for class 0 (not signed for term deposite) & column
# Probability of Successful deposite = 0.51484432
```

```
[[0.48515568 0.51484432]]
[1]
```

```
/home/volt/.local/lib/python3.10/site-packages/sklearn/base.py:409: UserWar
ning: X does not have valid feature names, but DecisionTreeClassifier was f
itted with feature names
  warnings.warn(
/home/volt/.local/lib/python3.10/site-packages/sklearn/base.py:409: UserWar
ning: X does not have valid feature names, but DecisionTreeClassifier was f
itted with feature names
  warnings.warn(
```

In [ ]:

```python
# Predict: Successful deposite with a maximun call duration = 3881 sec

print(dt2.predict_proba(np.array([0, 0, 3881, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,
print(dt2.predict(np.array([0, 0, 3881, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
[[0.19295499 0.80704501]]
[1]
```

```
/home/volt/.local/lib/python3.10/site-packages/sklearn/base.py:409: UserWar
ning: X does not have valid feature names, but DecisionTreeClassifier was f
itted with feature names
  warnings.warn(
/home/volt/.local/lib/python3.10/site-packages/sklearn/base.py:409: UserWar
ning: X does not have valid feature names, but DecisionTreeClassifier was f
itted with feature names
  warnings.warn(
```

In [ ]:

```python
# Get a row with poutcome_success = 1
#bank_with_dummies[(bank_with_dummies.poutcome_success == 1)]
data_drop_deposite.iloc[985]
```

Out[ ]:

```
age                      46.000000
balance                3354.000000
duration                522.000000
campaign                  1.000000
previous                  1.000000
default_cat               0.000000
housing_cat               1.000000
loan_cat                  0.000000
recent_pdays              0.005747
job_blue-collar           0.000000
job_entrepreneur          0.000000
job_other                 1.000000
job_pink-collar           0.000000
job_self-employed         0.000000
job_technician            0.000000
job_white-collar          0.000000
marital_divorced          1.000000
marital_married           0.000000
marital_single            0.000000
education_primary         0.000000
education_secondary       1.000000
education_tertiary        0.000000
education_unknown         0.000000
poutcome_failure          0.000000
poutcome_success          1.000000
poutcome_unknown          0.000000
Name: 985, dtype: float64
```

In [ ]:
```
# Predict: Probability for above

print(dt2.predict_proba(np.array([46,3354,522,1,1,0,1,0,0.005747,0,0,1,0,0,0
#print(ctree.predict(np.array([46,3354,522,1,1,0,1,0,0.005747,0,0,1,0,0,0,0,
```

```
[[0.19295499 0.80704501]]
```

```
/home/volt/.local/lib/python3.10/site-packages/sklearn/base.py:409: UserWar
ning: X does not have valid feature names, but DecisionTreeClassifier was f
itted with feature names
  warnings.warn(
```

In [ ]:
```
# Make predictions on the test set
preds = dt2.predict(data_test)

# Calculate accuracy
print("\nAccuracy score: \n{}".format(metrics.accuracy_score(label_test, pre

# Make predictions on the test set using predict_proba
probs = dt2.predict_proba(data_test)[:,1]

# Calculate the AUC metric
print("\nArea Under Curve: \n{}".format(metrics.roc_auc_score(label_test, pr
```

```
Accuracy score:
0.7268248992386923

Area Under Curve:
0.7880265888143609
```