In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import random
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
bank = pd.read_csv("C:/Users/sheri/Desktop/projectbank.csv")
```

In [3]:
```
bank.shape
```

Out[3]: (45211, 17)

In [4]:
```
bank.head()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	

In [5]:
```
bank.isna().sum()
```

Out[5]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

In [6]:
```
bank.dtypes
#There are numerical and categorical data.
```

Out[6]:
```
age           int64
```

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

In [7]:

```
bank.columns
```

Out[7]:

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

In [8]:

```
bank.median()
```

Out[8]:

```
age          39.0
balance     448.0
day          16.0
duration    180.0
campaign      2.0
pdays        -1.0
previous      0.0
dtype: float64
```

In [9]:

```
bank.mean()
```

Out[9]:

```
age           40.936210
balance     1362.272058
day           15.806419
duration     258.163080
campaign       2.763841
pdays         40.197828
previous       0.580323
dtype: float64
```

In [10]:

```
bank.min()
```

Out[10]:

```
age                18
job            admin.
marital      divorced
education     primary
default            no
balance         -8019
housing            no
loan               no
contact      cellular
day                 1
month             apr
```

```
duration            0
campaign            1
pdays              -1
previous            0
poutcome      failure
y                  no
dtype: object
```

In [11]:

```
bank.max()
```

Out[11]:

```
age               95
job          unknown
marital       single
education    unknown
default          yes
balance       102127
housing          yes
loan             yes
contact      unknown
day               31
month            sep
duration        4918
campaign          63
pdays            871
previous         275
poutcome     unknown
y                yes
dtype: object
```

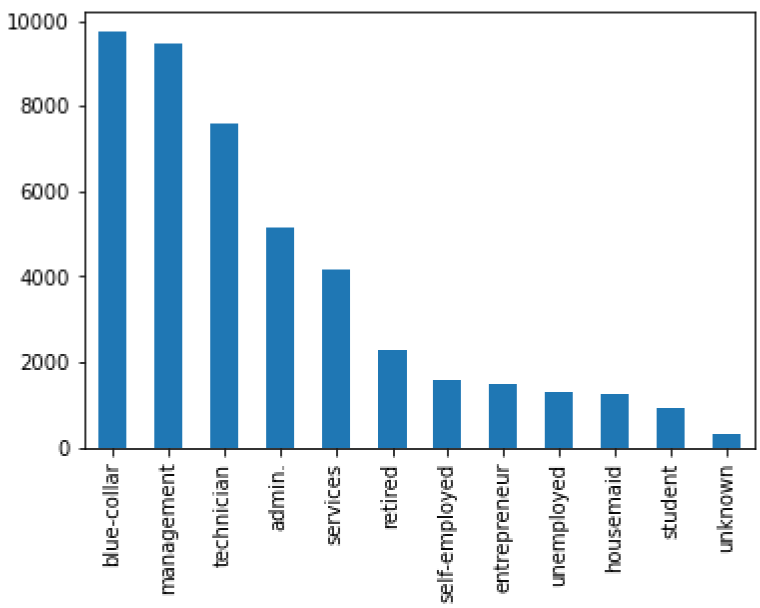
What simple models have you tried

In [12]:

```
df = bank.copy()
```

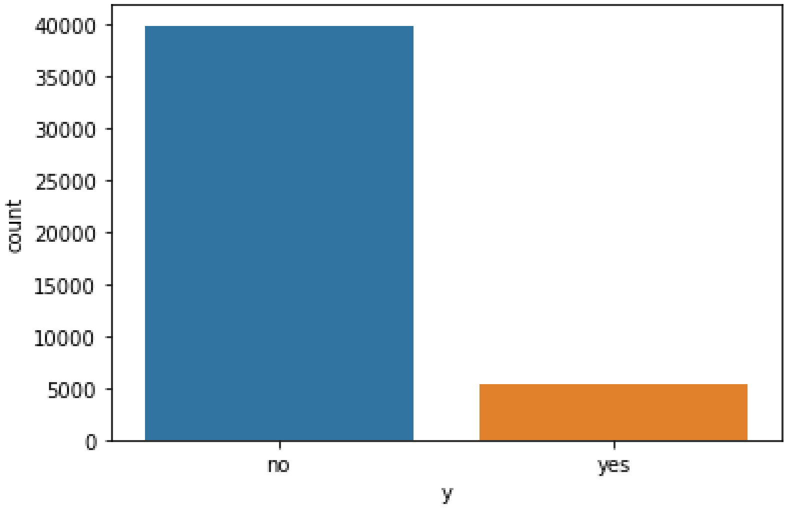
In [13]:

```
bank['job'].value_counts().plot(kind='bar');
```



In [14]:

```
ax = sns.countplot(x = df["y"])  #Imbalanced dataset
plt.show()
```



categorical variables = ["job", "marital", "education", "default", "housing", "loan", "contact", "month", "day_of_week", "poutcome"]

numerical variables = ["age", "duration", "campaign", "pdays", "previous", "emp.var.rate", "cons.price.idx", "cons.conf.idx", "euribor3m", "nr.employed"]

In [15]:

```
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

In [16]:

```
objfeatures = df.select_dtypes(include="object").columns
le = LabelEncoder()

for feat in objfeatures:
    df[feat] = le.fit_transform(df[feat].astype(str))
```

In [17]:

```
X = df.drop('y', 1)
y = df['y']

X = StandardScaler().fit_transform(X.astype(int))
```

In [18]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

In [19]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn import tree
```

In [20]:

```
short_tree = tree.DecisionTreeClassifier(max_depth = 3)
short_tree = short_tree.fit(X_train, y_train)
y_pred = short_tree.predict(X_test)
print('accuracy %2.2f ' % accuracy_score(y_test,y_pred))

cm = confusion_matrix(le.inverse_transform(y_test), le.inverse_transform(y_pred))
```

```
#test_results1 = pd.DataFrame(cm,index=labels,columns=labels)
#display(test_results1)#accuracy 0.66
```

```
accuracy 0.89
```

In [30]:

```
y.value_counts(normalize=True)
```

Out[30]:

```
0    0.883015
1    0.116985
Name: y, dtype: float64
```

Since the data is imbalanced f1 score or auc is a better metric to compare models

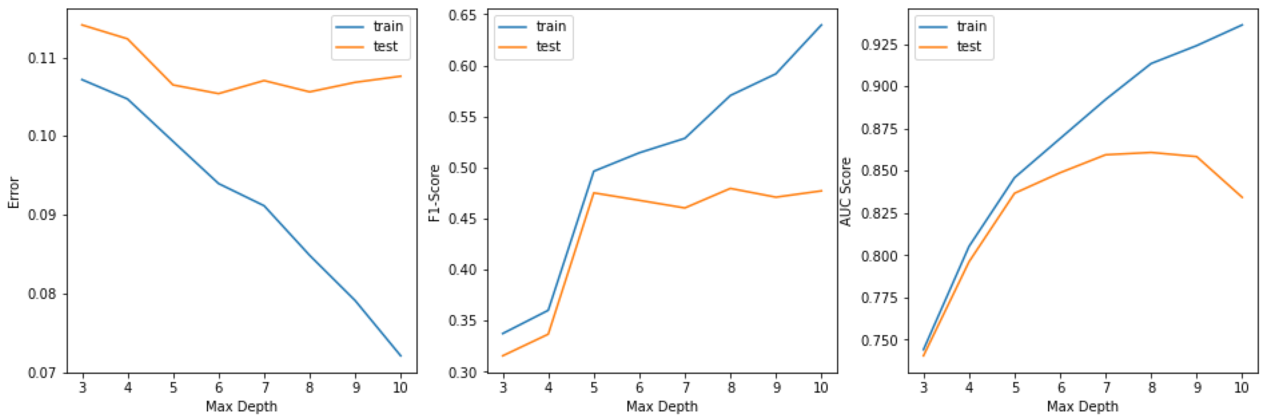
In [34]:

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
depths = []
accs = []
trainaccuracy = []
f1_train = []
f1_test = []
auc_train = []
auc_test = []
for i in range(3, 11):
    short_tree = tree.DecisionTreeClassifier(max_depth = i)
    short_tree = short_tree.fit(X_train, y_train)
    y_pred = short_tree.predict(X_test)
    y_prob = short_tree.predict_proba(X_test)[:,1]
    acc = accuracy_score(y_test, y_pred)
    train_pred = short_tree.predict(X_train)
    train_prob = short_tree.predict_proba(X_train)[:,1]
    auc_train.append(roc_auc_score(y_train, train_prob))
    auc_test.append(roc_auc_score(y_test, y_prob))
    f1_train.append(f1_score(y_train, train_pred))
    f1_test.append(f1_score(y_test, y_pred))
    trainacc = accuracy_score(y_train, train_pred)
    depths.append(i)
    accs.append(1-acc)
    trainaccuracy.append(1-trainacc)
```

In [37]:

```
plt.figure(figsize=(16,5))
plt.subplot(1,3,1)
plt.plot(depths,trainaccuracy, label = "train")
plt.plot(depths, accs, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('Error')
plt.legend()

plt.subplot(1,3,2)
plt.plot(depths,f1_train, label = "train")
plt.plot(depths, f1_test, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('F1-Score')
plt.legend()


plt.subplot(1,3,3)
```

```
plt.plot(depths,auc_train, label = "train")
plt.plot(depths, auc_test, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('AUC Score')
plt.legend()

plt.show()
```



In [38]:

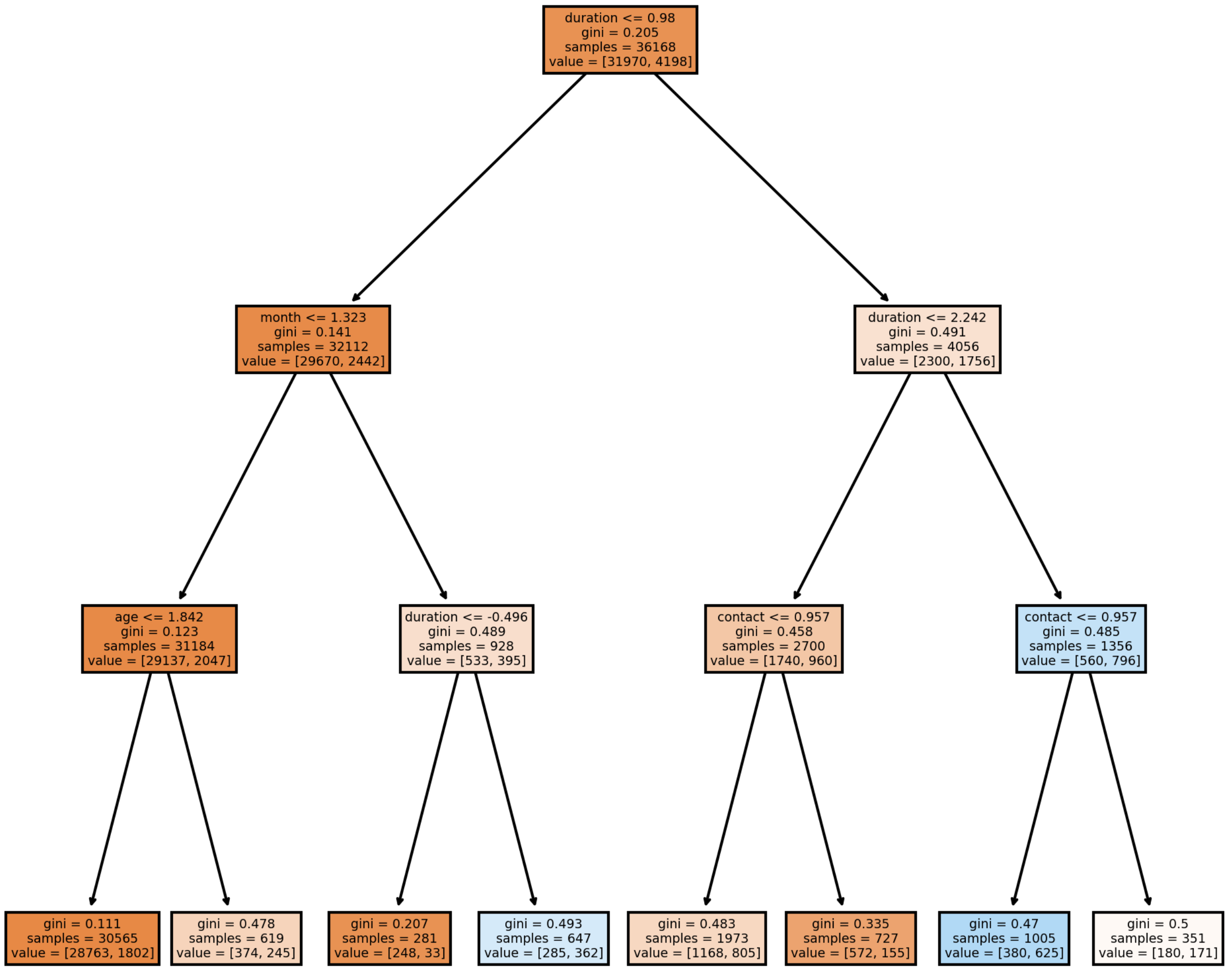
```
pd.DataFrame({'Depth': depths, 'Train_Error': trainaccuracy, 'Test_Error': accs,
              'Train_F1': f1_train, 'Test_F1': f1_test, 'Train_AUC': auc_train, 'Test_AU

#F1 score
```

Out[38]:

	Depth	Train_Error	Test_Error	Train_F1	Test_F1	Train_AUC	Test_AUC
0	3	0.107167	0.114121	0.337436	0.315650	0.744156	0.740511
1	4	0.104706	0.112352	0.360196	0.336815	0.805199	0.796087
2	5	0.099314	0.106491	0.496354	0.475204	0.845880	0.836646
3	6	0.093950	0.105385	0.514433	0.467895	0.869016	0.848791
4	7	0.091130	0.107044	0.528604	0.460424	0.892121	0.859422
5	8	0.084826	0.105607	0.570549	0.479564	0.913388	0.860801
6	9	0.079075	0.106823	0.591778	0.470975	0.923983	0.858337
7	10	0.072053	0.107597	0.639657	0.477163	0.936236	0.834246

In [45]:

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (8,8), dpi=300)
short_tree = tree.DecisionTreeClassifier(max_depth = 3)
short_tree = short_tree.fit(X_train, y_train)
tree.plot_tree(short_tree, filled=True, feature_names=df.drop(columns='y').columns);
```



Perceptron

In [46]:
```
from sklearn.linear_model import Perceptron
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X_train, y_train)
print("Acc:", clf.score(X_test, y_test))
```

```
Acc: 0.8458476169412805
```

In [61]:
```
from sklearn.linear_model import LogisticRegression
```

In [72]:
```
alphas = []
accs = []
trainaccuracy = []
f1_train = []
f1_test = []
auc_train = []
```

```
auc_test = []
for i in 10.0**np.arange(-7,2):
    clf = LogisticRegression(random_state=0,C=i,penalty='l2',  )
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_prob = clf.predict_proba(X_test)[:,1]
    acc = accuracy_score(y_test, y_pred)
    train_pred = clf.predict(X_train)
    train_prob = clf.predict_proba(X_train)[:,1]
    auc_train.append(roc_auc_score(y_train, train_prob))
    auc_test.append(roc_auc_score(y_test, y_prob))
    f1_train.append(f1_score(y_train, train_pred))
    f1_test.append(f1_score(y_test, y_pred))
    trainacc = accuracy_score(y_train, train_pred)
    alphas.append(i)
    accs.append(1-acc)
    trainaccuracy.append(1-trainacc)
```

In [73]:

```
pd.DataFrame({'alpha': alphas, 'Train_Error': trainaccuracy, 'Test_Error': accs,
             'Train_F1': f1_train, 'Test_F1': f1_test, 'Train_AUC': auc_train, 'Test_AU

#F1 score
```

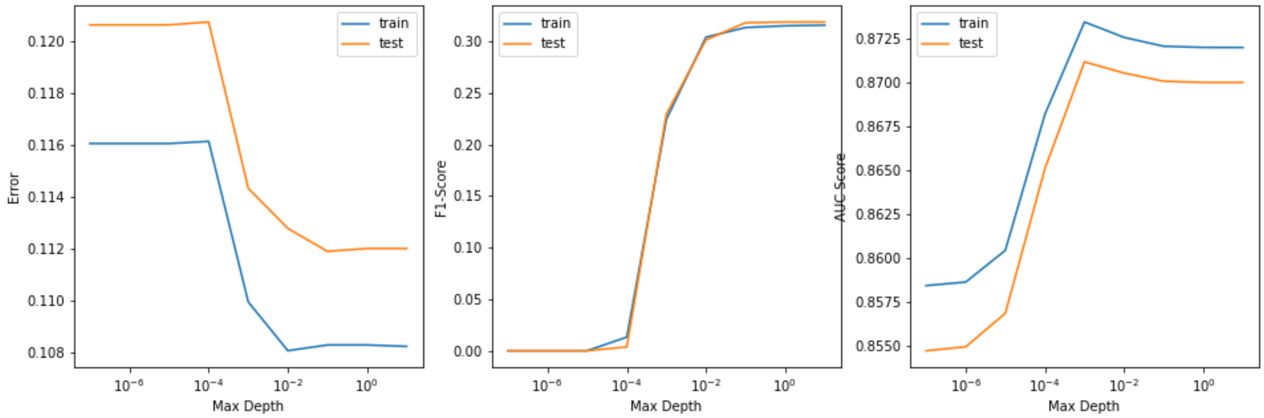
Out[73]:

	alpha	Train_Error	Test_Error	Train_F1	Test_F1	Train_AUC	Test_AUC
0	1.000000e-07	0.116069	0.120646	0.000000	0.000000	0.858413	0.854704
1	1.000000e-06	0.116069	0.120646	0.000000	0.000000	0.858614	0.854922
2	1.000000e-05	0.116069	0.120646	0.000000	0.000000	0.860421	0.856836
3	1.000000e-04	0.116152	0.120756	0.013155	0.003650	0.868168	0.865073
4	1.000000e-03	0.109959	0.114343	0.224303	0.229508	0.873432	0.871160
5	1.000000e-02	0.108079	0.112794	0.304077	0.301370	0.872555	0.870526
6	1.000000e-01	0.108300	0.111910	0.313409	0.318059	0.872050	0.870057
7	1.000000e+00	0.108300	0.112020	0.315090	0.318763	0.871985	0.869991
8	1.000000e+01	0.108245	0.112020	0.315679	0.318763	0.871979	0.869986

In [74]:

```
plt.figure(figsize=(16,5))
plt.subplot(1,3,1)
plt.plot(alphas,trainaccuracy, label = "train")
plt.plot(alphas, accs, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('Error')
plt.legend()
plt.xscale('log')

plt.subplot(1,3,2)
plt.plot(alphas,f1_train, label = "train")
plt.plot(alphas, f1_test, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('F1-Score')
plt.legend()
plt.xscale('log')
```

```
plt.subplot(1,3,3)
plt.plot(alphas,auc_train, label = "train")
plt.plot(alphas, auc_test, label = "test")
plt.xlabel('Max Depth')
plt.ylabel('AUC Score')
plt.legend()
plt.xscale('log')

plt.show()
```



In [ ]: