Targeting Direct Marketing

Background

Direct marketing, either through mail, email, phone, etc., is a common tactic to acquire customers. Because resources and a customer's attention are limited, the goal is to only target the subset of prospects who are likely to engage with a specific offer. Predicting those potential customers based on readily available information like demographics, past interactions, and environmental factors is a common machine learning problem.

This notebook presents an example problem to predict if a customer will enroll for a term deposit at a bank after one or more phone calls. The steps include:

- Preparing your Amazon SageMaker notebook
- Downloading data from the internet into Amazon SageMaker
- Investigating and transforming the data so that it can be fed to Amazon SageMaker algorithms
- Estimating a model
- Evaluating the effectiveness of the model
- Setting the model up to make ongoing predictions

Preparation

This notebook was created and tested on an ml.m4.xlarge notebook instance.

Start by specifying:

- The Amazon Simple Storage Service (Amazon S3) bucket and prefix that you want to use for training and model data. This should be within the same Region as the Notebook Instance, training, and hosting.
- The AWS Identity and Access Management (IAM) role to give training and hosting access to your data. See the documentation for how to create these.

Note: Here you will use the execution role the current notebook instance was given when it was created. This role has necessary permissions, including access to your data in S3. If you wish to use a different role, replace sagemaker.get_execution_role() with the appropriate IAM role or ARN.

Replace **`<LabBucketName>`** with the resource name that was provided with your lab account.

In [1]:
```
#bucket = 'ml-pipeline-bucket'
bucket = 'ml-pipeline-bucket'
prefix = 'sagemaker/DEMO-xgboost-dm'

# Define IAM role
import boto3
import re
from sagemaker import get_execution_role

role = get_execution_role()
```

Bring in the Python libraries that you'll use throughout the analysis.

In [2]:
```
import numpy as np                                # For matrix operations and
 numerical processing
import pandas as pd                               # For munging tabular data
import matplotlib.pyplot as plt                   # For charts and visualizati
ons
from IPython.display import Image                 # For displaying images in t
he notebook
from IPython.display import display               # For displaying outputs in
 the notebook
from time import gmtime, strftime                 # For labeling Amazon SageMa
ker models, endpoints, etc.
import sys                                        # For writing outputs to the
notebook
import math                                       # For ceiling function
import json                                       # For parsing hosting output
s
import os                                         # For manipulating filepath
 names
import sagemaker                                  # Amazon SageMaker's Python
 SDK provides many helper functions
from sagemaker.predictor import CSVSerializer    # Converts strings for HTTP P
OST requests on inference
```

Data

Start by downloading the [Bank Marketing Data Set (https://archive.ics.uci.edu/ml/datasets/bank+marketing)](https://archive.ics.uci.edu/ml/datasets/bank+marketing) from the University of California, Irvine (UCI) Machine Learning ML Repository.

Source information:

S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

In [3]:
```
!aws s3 cp s3://aws-tc-largeobjects/ILT-TF-200-MLDWTS/lab1/bank-additional.zip
.
#!wget https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-ad
ditional.zip
!unzip -o bank-additional.zip
```

```
download: s3://aws-tc-largeobjects/ILT-TF-200-MLDWTS/lab1/bank-additional.zip
to ./bank-additional.zip
Archive:  bank-additional.zip
   creating: bank-additional/
  inflating: bank-additional/.Rhistory
  inflating: bank-additional/.DS_Store
  inflating: bank-additional/bank-additional-names.txt
  inflating: bank-additional/bank-additional.csv
  inflating: bank-additional/bank-additional-full.csv
```

Now, read this into a Pandas dataframe and take a look.

In [4]:
```
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)     # Makes sure you can see all of
 the columns
pd.set_option('display.max_rows', 20)         # Keeps the output on one page
```

In [5]:
```
data.head()
```

Out[5]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon
1	57	services	married	high.school	unknown	no	no	telephone	may	mon
2	37	services	married	high.school	no	yes	no	telephone	may	mon
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon
4	56	services	married	high.school	no	no	yes	telephone	may	mon

Let's talk about the data. At a high level, you can see:

- There are just over 40K customer records and 20 features for each customer.
- The features are mixed--some numeric, some categorical.
- The data appears to be sorted, at least by `time` and `contact`.

Exploration

Let's explore the data. First, let's understand how the features are distributed.

In [6]:
```
# Frequency tables for each categorical feature
for column in data.select_dtypes(include=['object']).columns:
    display(pd.crosstab(index=data[column], columns='% observations', normaliz
e='columns')*100)
```

col_0 job	% observations
admin.	25.303486
blue-collar	22.467709
entrepreneur	3.535010
housemaid	2.573565
management	7.099155
retired	4.175974
self-employed	3.450034
services	9.636302
student	2.124405
technician	16.371273
unemployed	2.461882
unknown	0.801204

col_0 marital	% observations
divorced	11.197436
married	60.522482
single	28.085850
unknown	0.194231

col_0 education	% observations
basic.4y	10.138875
basic.6y	5.564728
basic.9y	14.676605
high.school	23.101389
illiterate	0.043702
professional.course	12.729436
university.degree	29.542585
unknown	4.202680

col_0 default	% observations
no	79.120132
unknown	20.872584
yes	0.007284

col_0 housing	% observations
no	45.212198
unknown	2.403613
yes	52.384190

col_0 loan	% observations
no	82.426920
unknown	2.403613
yes	15.169467

col_0 contact	% observations
cellular	63.474798
telephone	36.525202

col_0 month	% observations
apr	6.390211
aug	14.999514
dec	0.441876
jul	17.417694
jun	12.911528
mar	1.325629
may	33.429640
nov	9.956784
oct	1.743226
sep	1.383898

col_0	% observations
day_of_week	
fri	19.003108
mon	20.671069
thu	20.935709
tue	19.641643
wed	19.748470

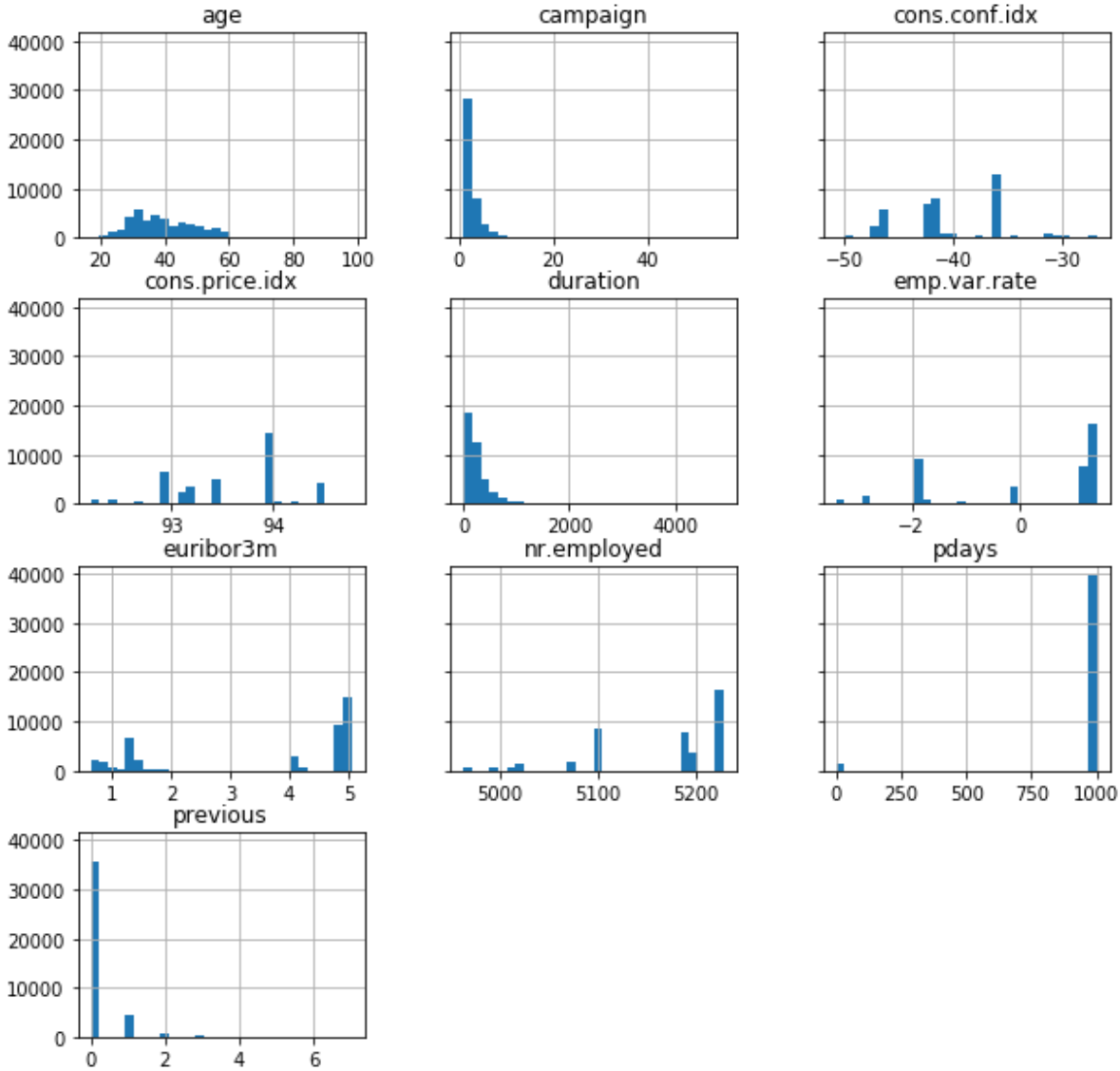
col_0	% observations
poutcome	
failure	10.323395
nonexistent	86.343110
success	3.333495

col_0	% observations
y	
no	88.734583
yes	11.265417

In [7]:
```
# Histograms for each numeric feature
display(data.describe())
display(data.describe(include=np.object))
#%matplotlib inline
hist = data.hist(bins=30, sharey=True, figsize=(10, 10))
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	4118
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	9
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	9
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	9
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	9
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	9
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	9

	job	marital	education	default	housing	loan	contact	month	day_of_week
count	41188	41188	41188	41188	41188	41188	41188	41188	41188
unique	12	4	8	3	3	3	2	10	5
top	admin.	married	university.degree	no	yes	no	cellular	may	thu
freq	10422	24928	12168	32588	21576	33950	26144	13769	8623


age
campaign
cons.conf.idx
cons.price.idx
duration
emp.var.rate
euribor3m
nr.employed
pdays
previous

Notice that:

- Almost 90% of the values for our target variable y are "no", so most customers did not subscribe to a term deposit.
- Many of the predictive features take on values of "unknown". Some are more common than others. We should think carefully as to what causes a value of "unknown" (are these customers non-representative in some way?) and how to handle that.
- Even if "unknown" is included as its own distinct category, what does it mean, given that those observations likely fall within one of the other categories of that feature?
- Many of the predictive features have categories with few observations in them. If we find a small category to be highly predictive of our target outcome, do we have enough evidence to make a generalization about that?
- Contact timing is particularly skewed--almost a third in May and less than 1% in December. What does this mean for predicting our target variable next December?
- There are no missing values in our numeric features, or missing values have already been imputed.
  - `pdays` takes a value near 1,000 for almost all customers. This is likely a placeholder value signifying no previous contact.
- Several numeric features have a long tail. Do we need to handle these few observations with extremely large values differently?
- Several numeric features (particularly the macroeconomic ones) occur in distinct buckets. Should these be treated as categorical?

Next, let's look at how our features relate to the target that we are attempting to predict.

In [8]:
```
for column in data.select_dtypes(include=['object']).columns:
    if column != 'y':
        display(pd.crosstab(index=data[column], columns=data['y'], normalize=
'columns'))

for column in data.select_dtypes(exclude=['object']).columns:
    print(column)
    hist = data[[column, 'y']].hist(by='y', bins=30)
    plt.show()
```

y job	no	yes
admin.	0.248167	0.291379
blue-collar	0.235745	0.137500
entrepreneur	0.036445	0.026724
housemaid	0.026103	0.022845
management	0.071030	0.070690
retired	0.035187	0.093534
self-employed	0.034804	0.032112
services	0.099759	0.069612
student	0.016417	0.059267
technician	0.164523	0.157328
unemployed	0.023804	0.031034
unknown	0.008017	0.007974

y marital	no	yes
divorced	0.113166	0.102586
married	0.612783	0.545690
single	0.272190	0.349138
unknown	0.001861	0.002586

y education	no	yes
basic.4y	0.102550	0.092241
basic.6y	0.057568	0.040517
basic.9y	0.152457	0.101940
high.school	0.232133	0.222198
illiterate	0.000383	0.000862
professional.course	0.127175	0.128233
university.degree	0.287239	0.359914
unknown	0.040495	0.054095

y default	no	yes
no	0.776814	0.904526
unknown	0.223104	0.095474
yes	0.000082	0.000000

y housing	no	yes
no	0.454088	0.436638
unknown	0.024160	0.023060
yes	0.521752	0.540302

y loan	no	yes
no	0.823574	0.829741
unknown	0.024160	0.023060
yes	0.152266	0.147198

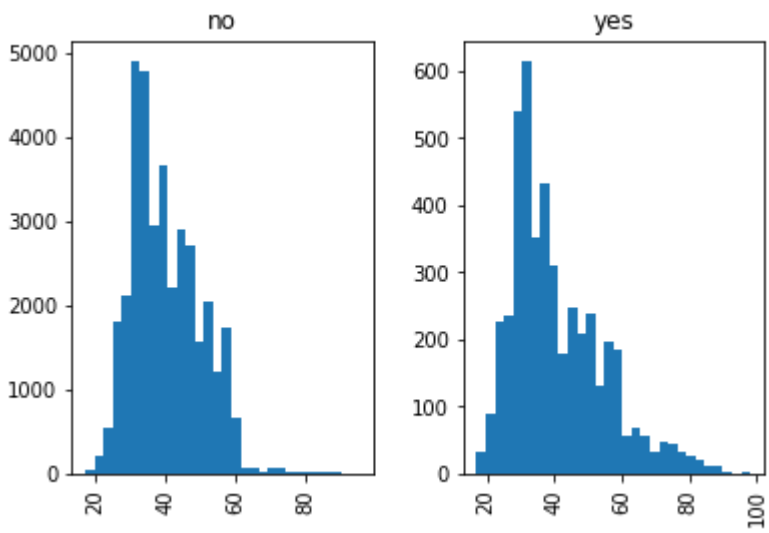
y contact	no	yes
cellular	0.60991	0.830388
telephone	0.39009	0.169612

y month	no	yes
apr	0.057267	0.116164
aug	0.151116	0.141164
dec	0.002545	0.019181
jul	0.178532	0.139871
jun	0.130212	0.120474
mar	0.007388	0.059483
may	0.352495	0.190948
nov	0.100826	0.089655
oct	0.011027	0.067888
sep	0.008591	0.055172

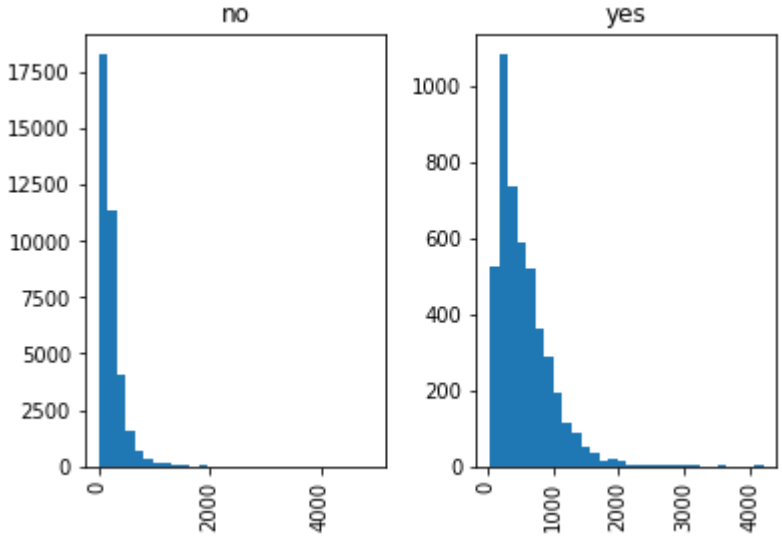
y	no	yes
day_of_week		
fri	0.191009	0.182328
mon	0.209779	0.182543
thu	0.207344	0.225216
tue	0.195277	0.205388
wed	0.196591	0.204526

y	no	yes
poutcome		
failure	0.099787	0.130388
nonexistent	0.887107	0.676940
success	0.013106	0.192672

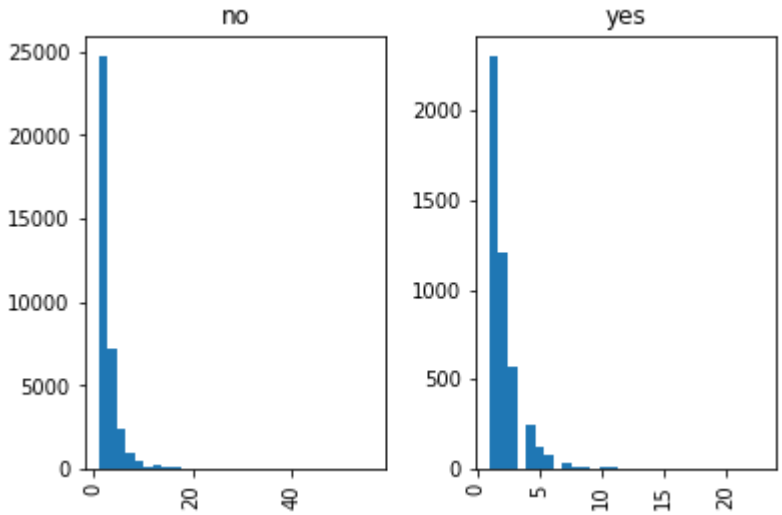
age



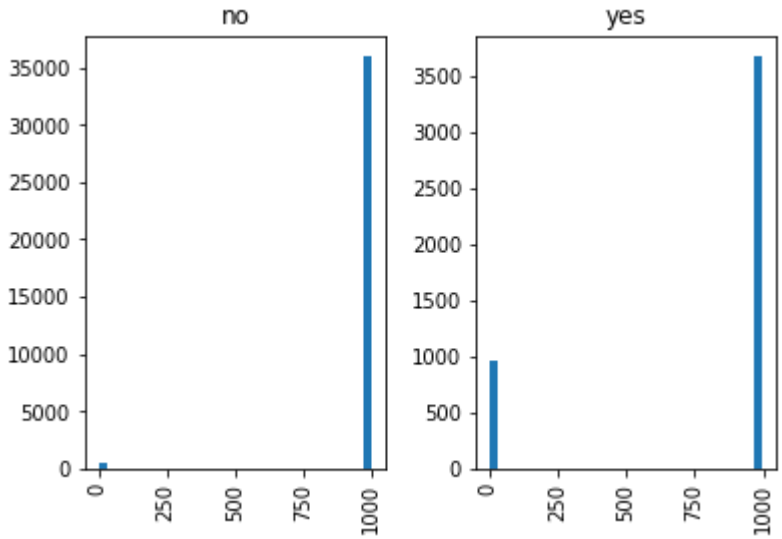
duration



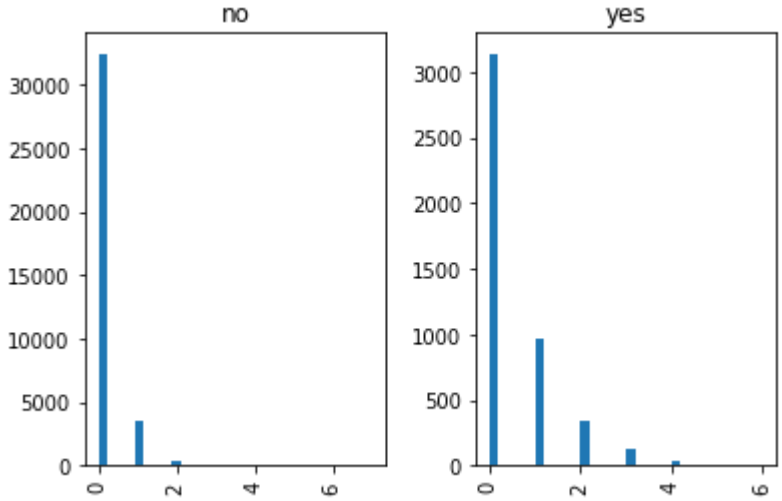

```
campaign
```



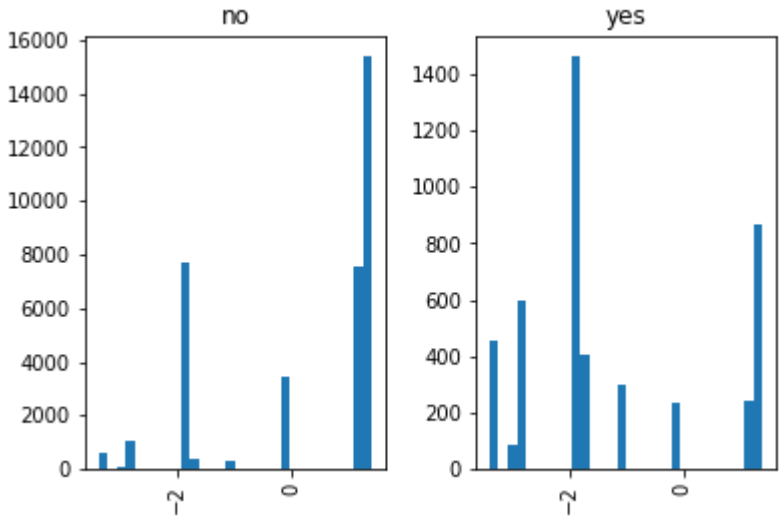

```
pdays
```



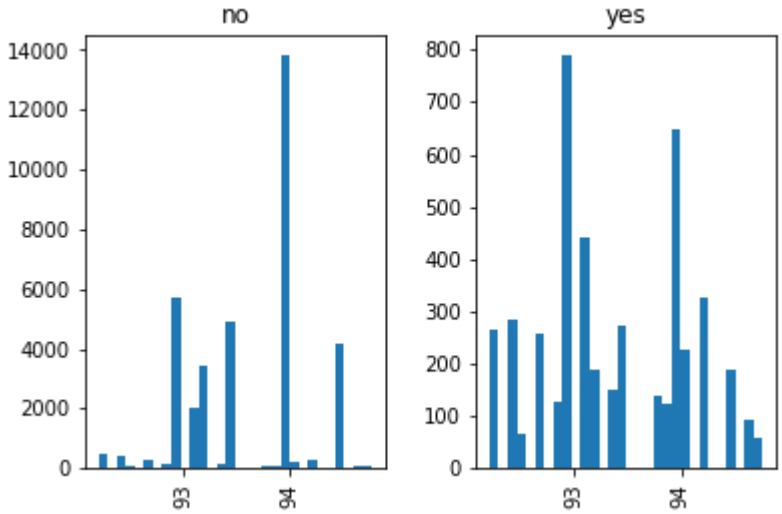

```
previous
```



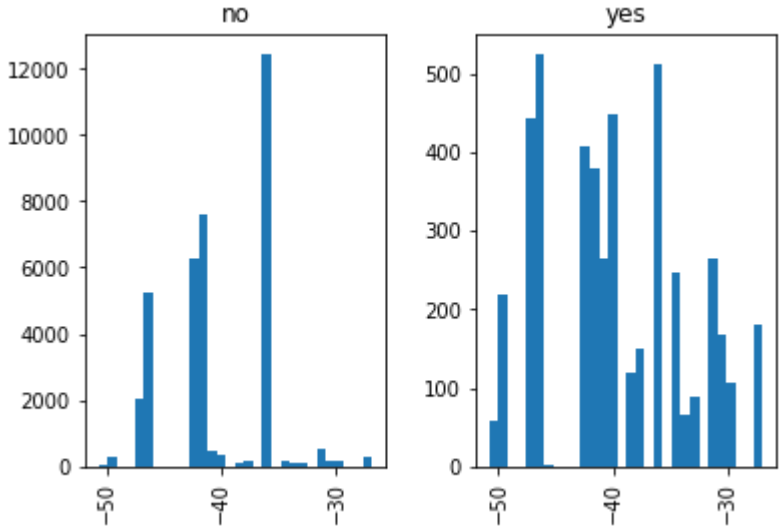
emp.var.rate



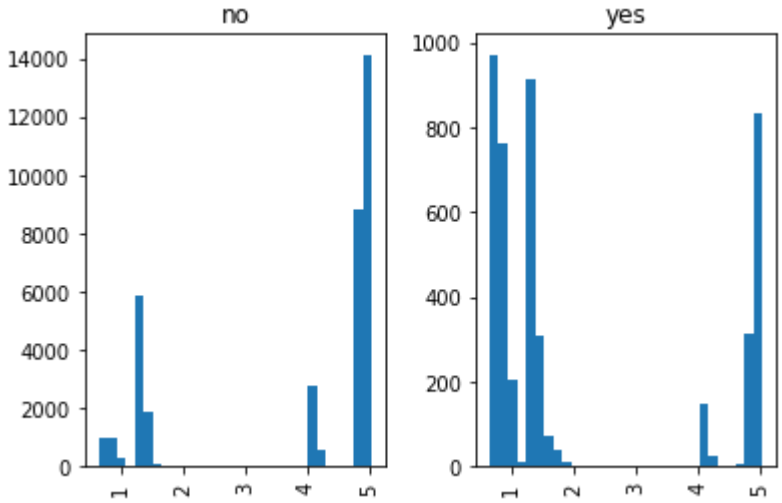
cons.price.idx



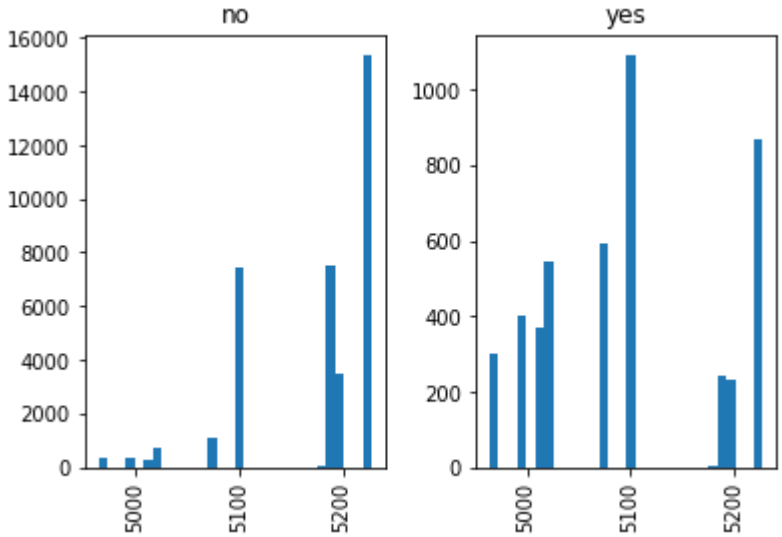
cons.conf.idx



euribor3m



nr.employed



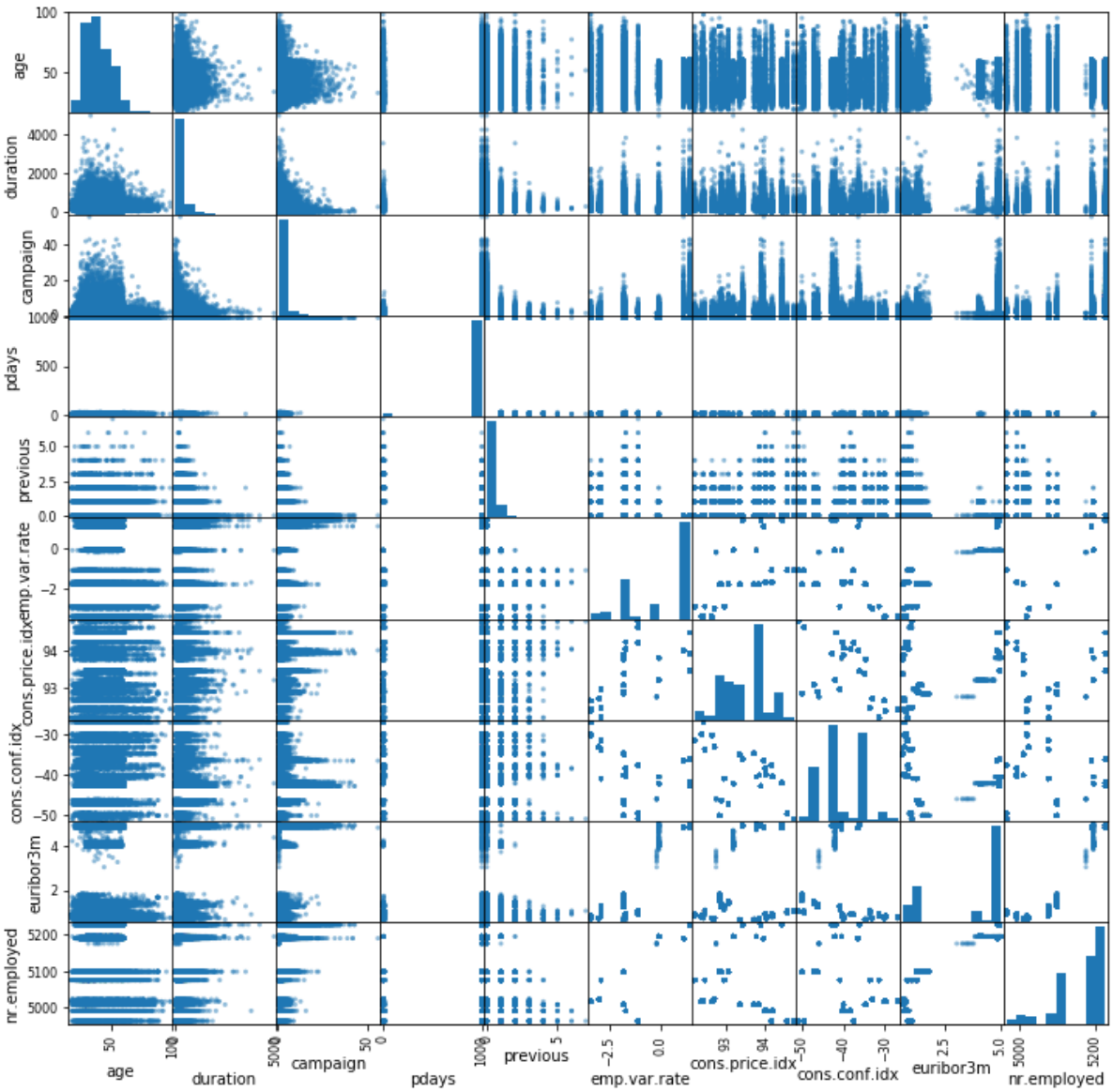
Notice that:

- Customers who are "blue-collar", "married", "unknown" default status, contacted by "telephone", and/or in "May" are a substantially lower portion of "yes" than "no" for subscribing.
- Distributions for numeric variables are different across "yes" and "no" subscribing groups, but the relationships may not be straightforward or obvious.

Now let's look at how our features relate to one another.

In [9]:
```
display(data.corr())
pd.plotting.scatter_matrix(data, figsize=(12, 12))
plt.show()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx
age	1.000000	-0.000866	0.004594	-0.034369	0.024365	-0.000371	0.000857
duration	-0.000866	1.000000	-0.071699	-0.047577	0.020640	-0.027968	0.005312
campaign	0.004594	-0.071699	1.000000	0.052584	-0.079141	0.150754	0.127836
pdays	-0.034369	-0.047577	0.052584	1.000000	-0.587514	0.271004	0.078889
previous	0.024365	0.020640	-0.079141	-0.587514	1.000000	-0.420489	-0.203130
emp.var.rate	-0.000371	-0.027968	0.150754	0.271004	-0.420489	1.000000	0.775334
cons.price.idx	0.000857	0.005312	0.127836	0.078889	-0.203130	0.775334	1.000000
cons.conf.idx	0.129372	-0.008173	-0.013733	-0.091342	-0.050936	0.196041	0.058986
euribor3m	0.010767	-0.032897	0.135133	0.296899	-0.454494	0.972245	0.688230
nr.employed	-0.017725	-0.044703	0.144095	0.372605	-0.501333	0.906970	0.522034



Notice that:

- Features vary widely in their relationships with one another. Some have highly negative correlation; others have highly positive correlation.
- Relationships between features are non-linear and discrete in many cases.

Transformation

Cleaning up data is part of nearly every machine learning project. It arguably presents the biggest risk if done incorrectly and is one of the more subjective aspects in the process.

In [10]:
```
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)
# Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'un
employed']), 1, 0)   # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)
# Convert categorical variables to sets of indicators
```

In [11]:
```
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx', 'c
ons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)
```

Train-test split

In [12]:
```
train_data, validation_data, test_data = np.split(model_data.sample(frac=1, ra
ndom_state=1729), [int(0.7 * len(model_data)), int(0.9 * len(model_data))])
# Randomly sort the data; then split out first 70%, second 20%, and last 10%
```

Amazon SageMaker's XGBoost container expects data in the libSVM or CSV data format. For this example, you'll stick to CSV. Note that the first column must be the target variable, and the CSV should not include headers. Also, notice that although repetitive, it's easiest to do this after the train|validation|test split rather than before. This avoids any misalignment issues due to random reordering.

In [13]:
```
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)], a
xis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], a
xis=1)], axis=1).to_csv('validation.csv', index=False, header=False)
```

Now, copy the file to Amazon S3 for Amazon SageMaker's managed training to pick up.

In [14]:
```
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'tra
in/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'val
idation/validation.csv')).upload_file('validation.csv')
```

Training

First, you need to specify the Amazon Elastic Container Registry (Amazon ECR) container location for Amazon SageMaker's implementation of XGBoost.

In [15]:
```
container = sagemaker.image_uris.retrieve('xgboost',boto3.Session().region_nam
e, '1.0-1')
```

Then, because you're training with the CSV file format, create `s3_input` s for the training function to use as a pointer to the files in Amazon S3. This also specifies that the content type is CSV.

In [16]:
```
s3_input_train = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/train'.for
mat(bucket, prefix), content_type='csv')
s3_input_validation = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/valid
ation/'.format(bucket, prefix), content_type='csv')
```

Now, you need to specify training parameters to the estimator. This includes:

- The `xgboost` algorithm container
- The IAM role to use
- Training instance type and count
- Amazon S3 location for output data
- Algorithm hyperparameters

And then a `.fit()` function, which specifies the Amazon S3 location for output data. In this case, both a training and validation set are passed in.

In [17]:
```
sess = sagemaker.Session()

xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    instance_count=1,
                                    instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(buc
ket, prefix),
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        num_round=100)

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

```
2020-11-11 14:35:33 Starting - Starting the training job...
2020-11-11 14:35:35 Starting - Launching requested ML instances......
2020-11-11 14:36:47 Starting - Preparing the instances for training......
2020-11-11 14:37:47 Downloading - Downloading input data...
2020-11-11 14:38:04 Training - Downloading the training image..INFO:sagemaker
-containers:Imported framework sagemaker_xgboost_container.training
INFO:sagemaker-containers:Failed to parse hyperparameter objective value bina
ry:logistic to Json.
Returning the value itself
INFO:sagemaker-containers:No GPUs detected (normal if no gpus installed)
INFO:sagemaker_xgboost_container.training:Running XGBoost Sagemaker in algori
thm mode
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
[14:38:44] 28831x59 matrix with 1701029 entries loaded from /opt/ml/input/dat
a/train?format=csv&label_column=0&delimiter=,
INFO:root:Determined delimiter of CSV input is ','
[14:38:44] 8238x59 matrix with 486042 entries loaded from /opt/ml/input/data/
validation?format=csv&label_column=0&delimiter=,
INFO:root:Single node training.
INFO:root:Train matrix has 28831 rows
INFO:root:Validation matrix has 8238 rows
[14:38:44] WARNING: /workspace/src/learner.cc:328:
Parameters: { num_round, silent } might not be used.

  This may not be accurate due to some parameters are only used in language b
indings but
  passed down to XGBoost core.  Or some parameters are not used but slip thro
ugh this
  verification. Please open an issue if you find above cases.


[0]#011train-error:0.10097#011validation-error:0.10682
[1]#011train-error:0.10007#011validation-error:0.10282
[2]#011train-error:0.09878#011validation-error:0.10549
[3]#011train-error:0.09927#011validation-error:0.10306
[4]#011train-error:0.09885#011validation-error:0.10330
[5]#011train-error:0.09923#011validation-error:0.10342
[6]#011train-error:0.09937#011validation-error:0.10318
[7]#011train-error:0.09948#011validation-error:0.10257
[8]#011train-error:0.09923#011validation-error:0.10282
[9]#011train-error:0.09923#011validation-error:0.10269
[10]#011train-error:0.09875#011validation-error:0.10269
[11]#011train-error:0.09844#011validation-error:0.10257
[12]#011train-error:0.09857#011validation-error:0.10306
[13]#011train-error:0.09809#011validation-error:0.10330
[14]#011train-error:0.09812#011validation-error:0.10294
[15]#011train-error:0.09819#011validation-error:0.10318
[16]#011train-error:0.09816#011validation-error:0.10245
[17]#011train-error:0.09816#011validation-error:0.10306
[18]#011train-error:0.09819#011validation-error:0.10282
[19]#011train-error:0.09830#011validation-error:0.10294
[20]#011train-error:0.09837#011validation-error:0.10269
[21]#011train-error:0.09788#011validation-error:0.10306
```

```
[22]#011train-error:0.09812#011validation-error:0.10294
[23]#011train-error:0.09764#011validation-error:0.10354
[24]#011train-error:0.09778#011validation-error:0.10391
[25]#011train-error:0.09760#011validation-error:0.10415
[26]#011train-error:0.09785#011validation-error:0.10439
[27]#011train-error:0.09767#011validation-error:0.10439
[28]#011train-error:0.09771#011validation-error:0.10464
[29]#011train-error:0.09781#011validation-error:0.10415
[30]#011train-error:0.09771#011validation-error:0.10500
[31]#011train-error:0.09767#011validation-error:0.10512
[32]#011train-error:0.09733#011validation-error:0.10500
[33]#011train-error:0.09722#011validation-error:0.10549
[34]#011train-error:0.09701#011validation-error:0.10488
[35]#011train-error:0.09698#011validation-error:0.10452
[36]#011train-error:0.09670#011validation-error:0.10488
[37]#011train-error:0.09656#011validation-error:0.10452
[38]#011train-error:0.09667#011validation-error:0.10476
[39]#011train-error:0.09646#011validation-error:0.10464
[40]#011train-error:0.09656#011validation-error:0.10452
[41]#011train-error:0.09670#011validation-error:0.10439
[42]#011train-error:0.09656#011validation-error:0.10476
[43]#011train-error:0.09667#011validation-error:0.10476
[44]#011train-error:0.09642#011validation-error:0.10500
[45]#011train-error:0.09611#011validation-error:0.10476
[46]#011train-error:0.09625#011validation-error:0.10439
[47]#011train-error:0.09622#011validation-error:0.10439
[48]#011train-error:0.09632#011validation-error:0.10452
[49]#011train-error:0.09632#011validation-error:0.10476
[50]#011train-error:0.09628#011validation-error:0.10488
[51]#011train-error:0.09622#011validation-error:0.10488
[52]#011train-error:0.09611#011validation-error:0.10464
[53]#011train-error:0.09608#011validation-error:0.10464
[54]#011train-error:0.09618#011validation-error:0.10464
[55]#011train-error:0.09601#011validation-error:0.10488
[56]#011train-error:0.09601#011validation-error:0.10452
[57]#011train-error:0.09608#011validation-error:0.10476
[58]#011train-error:0.09604#011validation-error:0.10464
[59]#011train-error:0.09611#011validation-error:0.10464
[60]#011train-error:0.09597#011validation-error:0.10452
[61]#011train-error:0.09580#011validation-error:0.10464
[62]#011train-error:0.09587#011validation-error:0.10452
[63]#011train-error:0.09590#011validation-error:0.10488
[64]#011train-error:0.09580#011validation-error:0.10476
[65]#011train-error:0.09583#011validation-error:0.10488
[66]#011train-error:0.09601#011validation-error:0.10500
[67]#011train-error:0.09580#011validation-error:0.10476
[68]#011train-error:0.09580#011validation-error:0.10512
[69]#011train-error:0.09573#011validation-error:0.10512
[70]#011train-error:0.09580#011validation-error:0.10500
[71]#011train-error:0.09587#011validation-error:0.10524
[72]#011train-error:0.09580#011validation-error:0.10549
[73]#011train-error:0.09570#011validation-error:0.10524
[74]#011train-error:0.09577#011validation-error:0.10561
[75]#011train-error:0.09566#011validation-error:0.10561
[76]#011train-error:0.09597#011validation-error:0.10561
[77]#011train-error:0.09594#011validation-error:0.10585
```

```
[78]#011train-error:0.09583#011validation-error:0.10585
[79]#011train-error:0.09580#011validation-error:0.10585
[80]#011train-error:0.09573#011validation-error:0.10597
[81]#011train-error:0.09563#011validation-error:0.10609
[82]#011train-error:0.09563#011validation-error:0.10609
[83]#011train-error:0.09563#011validation-error:0.10561
[84]#011train-error:0.09590#011validation-error:0.10549
[85]#011train-error:0.09580#011validation-error:0.10549
[86]#011train-error:0.09580#011validation-error:0.10537
[87]#011train-error:0.09563#011validation-error:0.10537
[88]#011train-error:0.09563#011validation-error:0.10537
[89]#011train-error:0.09563#011validation-error:0.10537
[90]#011train-error:0.09538#011validation-error:0.10561
[91]#011train-error:0.09535#011validation-error:0.10561
[92]#011train-error:0.09535#011validation-error:0.10549
[93]#011train-error:0.09552#011validation-error:0.10549
[94]#011train-error:0.09528#011validation-error:0.10585
[95]#011train-error:0.09531#011validation-error:0.10585
[96]#011train-error:0.09493#011validation-error:0.10585
[97]#011train-error:0.09500#011validation-error:0.10585
[98]#011train-error:0.09511#011validation-error:0.10573
[99]#011train-error:0.09517#011validation-error:0.10597

2020-11-11 14:39:00 Uploading - Uploading generated training model
2020-11-11 14:39:00 Completed - Training job completed
Training seconds: 73
Billable seconds: 73
```

Hosting

Now that you've trained the `xgboost` algorithm on the data, deploy a model that's hosted behind a real-time endpoint.

First, you need to determine how to pass data into and receive data from the endpoint. The data is currently stored as NumPy arrays in the memory of your notebook instance. To send it in an HTTP POST request, you'll serialize it as a CSV string and then decode the resulting CSV.

Note: For inference with the CSV format, Amazon SageMaker XGBoost requires that the data does NOT include the target variable.

In [18]:
```
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           serializer=CSVSerializer(),
                           instance_type='ml.m4.xlarge')
```

```
-------------!
```

Evaluation

There are many ways to compare the performance of a machine learning model, but start by simply comparing actual to predicted values. In this case, you're simply predicting whether the customer subscribed to a term deposit (`1`) or not (`0`), which produces a simple confusion matrix.

Now, use a simple function to:

1. Loop over the test dataset
2. Split it into mini-batches of rows
3. Convert those mini-batches to CSV string payloads (notice that the target variable is dropped from the dataset first)
4. Retrieve mini-batch predictions by invoking the XGBoost endpoint
5. Collect predictions and convert from the CSV output that the model provides into a NumPy array

In [19]:
```
def predict(data, rows=500):
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
    predictions = ''
    for array in split_array:
        predictions = ','.join([predictions, xgb_predictor.predict(array).deco
de('utf-8')])

    return np.fromstring(predictions[1:], sep=',')

predictions = predict(test_data.drop(['y_no', 'y_yes'], axis=1).values)
```

Now, check the confusion matrix to see how well the model predicted vs. actuals.

In [20]:
```
pd.crosstab(index=test_data['y_yes'], columns=np.round(predictions), rownames=
['actuals'], colnames=['predictions'])
```

Out[20]:

predictions	0.0	1.0
actuals		
0	3587	49
1	387	96

Of about 4,000 potential customers, the model predicted that 136 would subscribe, and 94 actually did. There were also 389 customers who subscribed that were not predicted to subscribe. This is less than desirable, but the model can (and should) be tuned to improve this. Most importantly, note that with minimal effort, the model produced accuracies similar to those published here (http://media.salford-systems.com/video/tutorial/2015/targeted_marketing.pdf).

Note: Because there is some element of randomness in the algorithm's subsample, your results may differ slightly from the text written above.

Cleanup

If you are finished with this notebook, run the following cell. This removes the hosted endpoint you created and avoids any charges from a stray instance being left on.

In [21]:
```
sagemaker.Session().delete_endpoint(xgb_predictor.endpoint_name)
```

In [ ]:
```
```