Project on term deposit prediction

In [1]:
```python
import pandas as pd
import numpy as np                    # For mathematical calculations
import seaborn as sns                 # For data visualization
import matplotlib.pyplot as plt
import seaborn as sn                  # For plotting graphs
%matplotlib inline
import warnings                       # To ignore any warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
# loading the data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [3]:
```python
train.columns
```

Out[3]:
```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'subscribed'],
      dtype='object')
```

In [4]:
```python
test.columns
```

Out[4]:
```
Index(['ID', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome'],
      dtype='object')
```

In [5]:
```python
train.shape, test.shape
```

Out[5]:
```
((31647, 18), (13564, 17))
```

We have 17 independent variables and 1 target variable, i.e. subscribed in the train dataset. We have similar features in the test dataset as the train dataset except the subscribed. We will predict the subscribed with the help of model built using the train data.

Next, let's look at how many categorical and numerical variables are there in our dataset. We will look at their data types.

In [6]:
```python
train.dtypes
```

Out[6]:
```
ID             int64
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
subscribed    object
dtype: object
```

In [7]:
```python
train.head()
```

Out[7]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previc
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	

Analysis

In [8]:
```python
from pandas_profiling import ProfileReport
FDprofile = ProfileReport(train)
FDprofile.to_file(output_file="fdprofile.html")
```

In [9]:
```python
train['subscribed'].value_counts()
```

Out[9]:
```
no     27932
yes     3715
Name: subscribed, dtype: int64
```

In [10]:
```python
train['subscribed'].value_counts(normalize=True)
```

Out[10]:
```
no     0.882611
yes    0.117389
Name: subscribed, dtype: float64
```

In [11]:
```python
sn.distplot(train["age"])
```

Out[11]:
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```

In [12]:
```python
train['job'].value_counts().plot.bar()
```

Out[12]:
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```

In [13]:
```python
train['default'].value_counts().plot.bar()
```

Out[13]:
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```

More than 90% of the clients have no default history. Now we will explore these variables against the target variable using bivariate analysis. We will make use of scatter plots for continuous or numeric variables and crosstabs for the categorical variables. Let's start with job and subscribed variable.

Bivariate

In [14]:
```python
print(pd.crosstab(train['job'],train['subscribed']))

job=pd.crosstab(train['job'],train['subscribed'])
job.div(job.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))
plt.xlabel('Job')
plt.ylabel('Percentage')
```

```
subscribed       no  yes
job
admin.         3179  452
blue-collar    6353  489
entrepreneur    923   85
housemaid       795   79
management     5716  923
retired        1212  362
self-employed   983  140
services       2649  254
student         453  182
technician     4713  594
unemployed      776  129
unknown         180   26
```

Out[14]:
```
Text(0, 0.5, 'Percentage')
```

From the above graph we can infer that students and retired people have higher chances of subscribing to a term deposit, which is surprising as students generally do not subscribe to a term deposit. The possible reason is that the number of students in the dataset is less and comparatively to other job types, more students have subscribed to a term deposit.

In [15]:
```python
print(pd.crosstab(train['default'],train['subscribed']))

default=pd.crosstab(train['default'],train['subscribed'])
default.div(default.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(8,8))
plt.xlabel('default')
plt.ylabel('Percentage')
```

```
subscribed     no   yes
default
no          27388  3674
yes           544    41
```

Out[15]:
```
Text(0, 0.5, 'Percentage')
```

We can infer that clients having no previous default have slightly higher chances of subscribing to a term loan as compared to the clients who have previous default history.

In [16]:
```python
train['subscribed'].replace('no', 0,inplace=True)
train['subscribed'].replace('yes', 1,inplace=True)
```

In [17]:
```python
corr = train.corr()
mask = np.array(corr)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corr, mask=mask,vmax=.9, square=True,annot=True, cmap="YlGnBu")
```

Out[17]:
```
<AxesSubplot:>
```

In [18]:
```python
train.isnull().sum()
```

Out[18]:
```
ID            0
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
subscribed    0
dtype: int64
```

There are no missing values in the train dataset.

Model Building

In [19]:
```python
target = train['subscribed']
train = train.drop('subscribed',1)
```

In [20]:
```python
train
```

Out[20]:

	ID	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	p
0	26110	56	admin.	married	unknown	no	1933	no	no	telephone	19	nov	44	2	-1	
1	40576	31	unknown	married	secondary	no	3	no	no	cellular	20	jul	91	2	-1	
2	15320	27	services	married	secondary	no	891	yes	no	cellular	18	jul	240	1	-1	
3	43962	57	management	divorced	tertiary	no	3287	no	no	cellular	22	jun	867	1	84	
4	29842	31	technician	married	secondary	no	119	yes	no	cellular	4	feb	380	1	-1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
31642	36483	29	management	single	tertiary	no	0	yes	no	cellular	12	may	116	2	-1	
31643	40178	53	management	divorced	tertiary	no	380	no	yes	cellular	5	jun	438	2	-1	
31644	19710	32	management	single	tertiary	no	312	no	no	cellular	7	aug	37	3	-1	
31645	38556	57	technician	married	secondary	no	225	yes	no	telephone	15	may	22	7	337	
31646	14156	55	management	divorced	secondary	no	204	yes	no	cellular	11	jul	1973	2	-1	

31647 rows × 17 columns

In [21]:
```python
train = pd.get_dummies(train)
```

In [22]:
```python
train
```

Out[22]:

	ID	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	mont
0	26110	56	1933	19	44	2	-1	0	1	0	...	0	0	0	
1	40576	31	3	20	91	2	-1	0	0	0	...	0	0	0	
2	15320	27	891	18	240	1	-1	0	0	0	...	0	0	0	
3	43962	57	3287	22	867	1	84	3	0	0	...	1	0	0	
4	29842	31	119	4	380	1	-1	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
31642	36483	29	0	12	116	2	-1	0	0	0	...	0	0	1	
31643	40178	53	380	5	438	2	-1	0	0	0	...	1	0	0	
31644	19710	32	312	7	37	3	-1	0	0	0	...	0	0	0	
31645	38556	57	225	15	22	7	337	12	0	0	...	0	0	1	
31646	14156	55	204	11	1973	2	-1	0	0	0	...	0	0	0	

31647 rows × 52 columns

In [23]:
```python
from sklearn.model_selection import train_test_split
```

In [24]:
```python
# splitting into train and validation with 20% data in validation set and 80% data in train set.
X_train, X_val, y_train, y_val = train_test_split(train, target, test_size = 0.2, random_state=12)
```

Logistic Regression

In [25]:
```python
from sklearn.linear_model import LogisticRegression
```

In [26]:
```python
lreg = LogisticRegression()
```

In [27]:
```python
lreg.fit(X_train,y_train)
```

Out[27]:
```
LogisticRegression()
```

In [28]:
```python
# making prediction on the validation set
prediction = lreg.predict(X_val)
```

Now we will evaluate how accurate our predictions are. As the evaluation metric for this problem is accuracy

In [29]:
```python
from sklearn.metrics import accuracy_score
```

In [30]:
```python
# calculating the accuracy score
accuracy_score(y_val, prediction)
```

Out[30]:
```
0.8881516587677725
```

Decision tree

In [31]:
```python
from sklearn.tree import DecisionTreeClassifier
```

In [32]:
```python
# defining the decision tree model with depth of 4, you can tune it further to improve the accuracy score
clf = DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [33]:
```python
# fitting the decision tree model
clf.fit(X_train,y_train)
```

Out[33]:
```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

In [34]:
```python
# making prediction on the validation set
predict = clf.predict(X_val)
```

In [35]:
```python
# calculating the accuracy score
accuracy_score(y_val, predict)
```

Out[35]:
```
0.9042654028436019
```

We got an accuracy of more than 90% on the validation set. You can try to improve the score by tuning hyperparameters of the model

In [36]:
```python
test = pd.get_dummies(test)
```

In [37]:
```python
test_prediction = clf.predict(test)
```

Finally, we will save these predictions into a csv file. You can then open this csv file and copy paste the predictions on the provided excel file to generate score.

In [38]:
```python
submission = pd.DataFrame()
```

In [39]:
```python
# creating a Business_Sourced column and saving the predictions in it
submission['ID'] = test['ID']
submission['subscribed'] = test_prediction
```

Since the target variable is yes or no, we will convert 1 and 0 in the predictions to yes and no respectively.

In [40]:
```python
submission['subscribed'].replace(0,'no',inplace=True)
submission['subscribed'].replace(1,'yes',inplace=True)
```

In [41]:
```python
submission.to_csv('submission.csv', header=True, index=False)
```

In [ ]: