Our goal is to build a model that predicts if the client will subscribe a term deposit.

Step 1: Load the dataset

In [1]:
```
import pandas as pd
df = pd.read_csv('bank.csv')
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.9+ MB
```

Here we can see that a lot of columns are shown as object data type, which means categorical. Some categorical columns need to be dropped and some categorical columns can be convert to boolean type.

Step 2: Data cleaning.

We need to replace categorical columns with boolean or numerical values in order to use them in building the model.

In [2]:
```
#Drop categorical variables
df = df.drop(['job', 'marital', 'education'], axis=1)
#Replace categorical columns with boolean or numerical values
df = df.replace({'default': {'yes': 1, 'no': 0, 'unknown': 2}})
df = df.replace({'housing': {'yes': 1, 'no': 0, 'unknown': 2}})
df = df.replace({'loan': {'yes': 1, 'no': 0, 'unknown': 2}})
df = df.replace({'contact': {'cellular': 1, 'telephone': 0}})
df = df.replace({'month': {'jan':1, 'feb':2, 'mar':3, 'apr':4, 'may':5,
'jun':6, 'jul':7, 'aug':8, 'sep':9, 'oct':10, 'nov':11, 'dec':12}})
df = df.replace({'day_of_week': {'mon':1, 'tue':2, 'wed':3, 'thu':4, 'fr
i':5, 'sat':6, 'sun':7}})
df = df.replace({'poutcome': {'success': 1, 'failure': 0, 'nonexistent':
2}})
df = df.replace({'y': {'yes': 1, 'no': 0}})
df.head()
```

Out[2]:

	age	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previc
0	56	0	0	0	0	5	1	261	1	999	
1	57	2	0	0	0	5	1	149	1	999	
2	37	0	1	0	0	5	1	226	1	999	
3	40	0	0	0	0	5	1	151	1	999	
4	56	0	0	1	0	5	1	307	1	999	

Now we have all columns with numerical values and ready to create the model.

Step 3: Split the dataset into 2 parts, 60% training data, 40% validation data.

In [3]:
```
len(df)
```

Out[3]: 41188

Right now, the whole dataset contains 41188 observations. We can put 24712 observations into training data, and 16476 observations into validation data.

In [4]:
```
import numpy as np
rows_for_training = np.random.choice( df.index, 24712, False )
training = df.index.isin(rows_for_training)
df_training = df[training]
df_validation = df[~training]
len(df_training), len(df_validation)
```

Out[4]:
```
(24712, 16476)
```

This code is being created by first generating 24712 random rows to df_training dataframe. And the rest will be put into df_validation dataframe. This gives us about 60% training set and 40% validation set.

Step 4: Create a model using the training dataset using the scikit-learn's logistic regression tool. And test the model using the validation dataset.

In [5]:
```
from sklearn.linear_model import LogisticRegression
def fit_model_to (training):
    predictors = training.iloc[:,:-1]
    response = training.iloc[:,-1]
    model = LogisticRegression()
    model.fit(predictors, response)
    return model

def score_model (M, validation):
    predictions = M.predict(validation.iloc[:,:-1] )
    TP = (validation['y'] & predictions).sum()
    FP = (~validation['y'] & predictions).sum()
    FN = (validation['y'] & ~predictions).sum()
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    return 2 * precision * recall / (precision + recall)

model = fit_model_to(df_training)
score_model(model, df_training), score_model(model, df_validation)
```

```
/opt/venv/lib/python3.7/site-packages/sklearn/linear_model/_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[5]:
```
(0.4971327745919718, 0.5019011406844106)
```

The values retured are the F1 score from training dataset and validation dataset. F1 score represents how good our model is by using the precision and recall values. We can see that both F1 score for training and validation datasets are relatively low. Possible reason is that we have high number of predictor variables. It's likely that some of predictors do not predict our response variable well.

Step 5: Create a better model to predict the response variable.

In [6]:
```
def fit_model_to (training):
    # fit the model the same way as step 4
    predictors = training.iloc[:,:-1]
    response = training.iloc[:,-1]
    model = LogisticRegression()
    model.fit(predictors, response)
    # fit another model to standardized predictors in order to compare t
he coefficients with the same scale
    standardized = (predictors - predictors.mean()) / predictors.std()
    standard_model = LogisticRegression()
    standard_model.fit(standardized, response)
    # get that model's coefficients and display them
    coeffs = pd.Series(standard_model.coef_[0], index=predictors.columns
)
    sorted = np.abs(coeffs).sort_values( ascending=False )  # sort the c
oefficients from the most important to the least
    coeffs = coeffs.loc[sorted.index]
    print(coeffs)
    # return the model fit to the actual predictors
    return model


model = fit_model_to(df_training)
print(score_model(model, df_training), score_model(model, df_validation
))
```

```
/opt/venv/lib/python3.7/site-packages/sklearn/linear_model/_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
emp.var.rate     -1.344532
duration          1.152690
cons.price.idx    0.587667
euribor3m         0.558025
nr.employed      -0.538965
contact           0.466749
pdays            -0.306265
cons.conf.idx     0.201882
default          -0.186847
poutcome          0.178152
month            -0.089427
campaign         -0.081138
age               0.036962
previous         -0.021448
loan             -0.021247
housing           0.020483
day_of_week       0.002305
dtype: float64
0.4971327745919718 0.5019011406844106
```

Here the coefficients are sorted in order of their importance to predict the model. We can perform a series of trial and error to see which columns to keep in the modeling in order to perform a higher F1 score.

In [178]:
```
columns = ['emp.var.rate', 'duration', 'cons.price.idx','nr.employed',
'contact','pdays','cons.conf.idx','default','previous','y']
model = fit_model_to( df_training.loc[:,columns] )
score_model( model, df_training.loc[:,columns] ), score_model( model, df
_validation.loc[:,columns] )
```

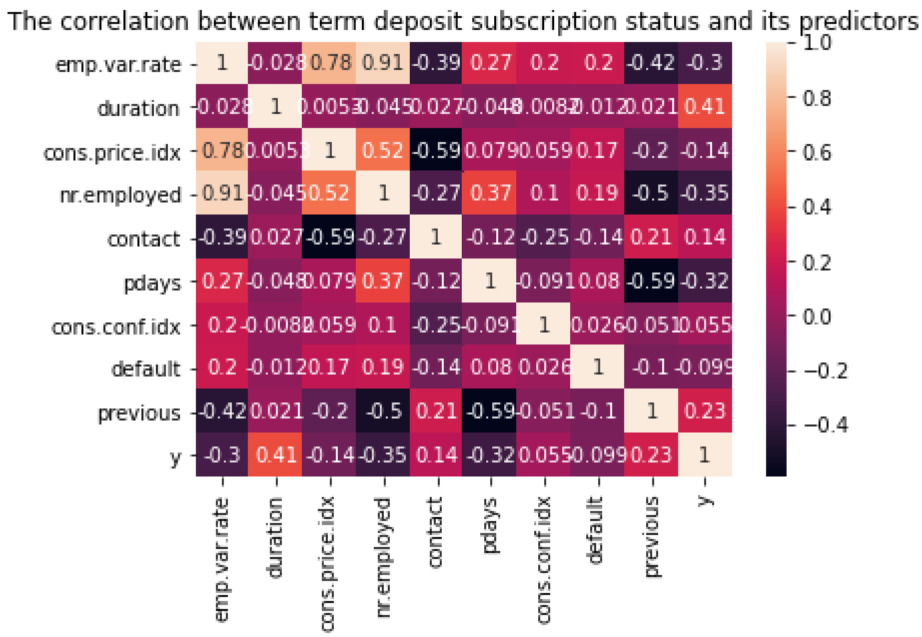
```
/opt/venv/lib/python3.7/site-packages/sklearn/linear_model/_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
emp.var.rate     -1.159004
duration          1.150195
cons.price.idx    0.679886
contact           0.431278
pdays            -0.340701
cons.conf.idx     0.250477
nr.employed      -0.238953
default          -0.176395
previous         -0.159186
dtype: float64
```

Out[178]:
```
(0.5070298769771528, 0.5085557837097878)
```

Here are the final predictors that we decided to keep after experimenting various combinations.

Step 6: Create a visualization between the predictors and repsonse variable

In [174]:
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
df_final = df.loc[:, columns]
correlation_coefficients = np.corrcoef(df_final, rowvar=False )
sns.heatmap( correlation_coefficients, annot=True )
plt.yticks( np.arange(10)+0.5, df_final.columns, rotation=0 )
plt.xticks( np.arange(10)+0.5, df_final.columns, rotation=90 )
plt.title('The correlation between term deposit subscription status and
 its predictors')
plt.show()
```



This is the heatmap showing the correlation between term deposit subscription status and its predictors, shown on the last column of the heatmap. We can see that none of the predictors is highly correlated with the response variable, which explains relatively low F1 score from the logistic regression modeling. Another notice is that there is high correlation between employee variation rate and number of employed. We decided to keep both variables, because the F1 score is about 2% higher than keeping just one.