Bank Marketing Analysis

The data set used here is from UCI machine learning repository. It is derived from the direct marketing campaigns of a Portuguese banking institution. The question is how to improve marketing campaign of a bank and know which customers are more likely to subscribe the bank’s product of term deposit or not.

Built 2 main models in Python:

- Decision Tree Classifier
- Logistic Regression

In [1]:
```
import numpy as np
import pandas as pd
```

1. Loading the Dataset

The data file is a newest update, named as bank_newest.csv, with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010). It is downloaded and renamed from the original file 'bank-additional-full.csv' in the sourse https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing)

It related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

In [2]:
```python
df = pd.read_csv('./bank/bank_newest.csv', sep=';')
df.head()
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.ra
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1

5 rows × 21 columns

Input variables:

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed) - *Note more:* we will replace ['basic.6y','basic.4y', 'basic.9y'] with 'basic' to make categorical simple
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric). *Important note:* this attribute highly affects the output target (e.g., if duration=0 then y='no')
12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):
21 - y - has the client subscribed a term deposit? ('yes','no')

In [3]:
```
df.describe()
```

Out[3]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

2. Feature Engineering / Data Preparation

2.1. Cleaning the Data (Pre Processing)

In [4]:
```
# We will not need to several features for analysis, so we can clean it.
df.drop(['duration','contact','month','day_of_week','pdays'],axis=1,inplace=True)
```

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

Out[5]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
0	56	housemaid	married	basic.4y	no	no	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
1	57	services	married	high.school	unknown	no	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
2	37	services	married	high.school	no	yes	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
3	40	admin.	married	basic.6y	no	no	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
4	56	services	married	high.school	no	no	yes	1	0	nonexistent	1.1	93.994	-36.4	4.857

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
campaign          0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

There is no null data, so we don't need to drop it.

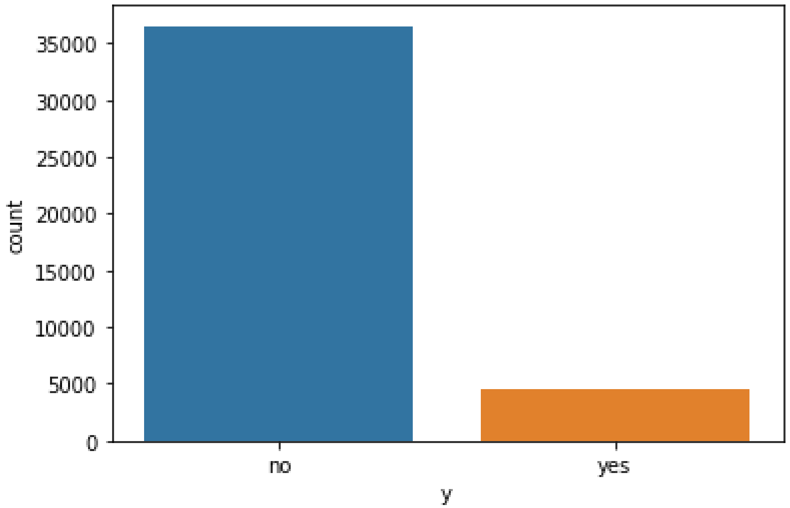
In [7]:
```
# We replace ['basic.6y','basic.4y', 'basic.9y'] with 'basic' to make categorical simple for feature engineering
df.replace(['basic.6y','basic.4y', 'basic.9y'], 'basic', inplace=True)
```

2.2 Visualizing the Data

In [8]:
```
import seaborn as sns
```

In [9]:
```
sns.countplot(x='y', data=df)
```

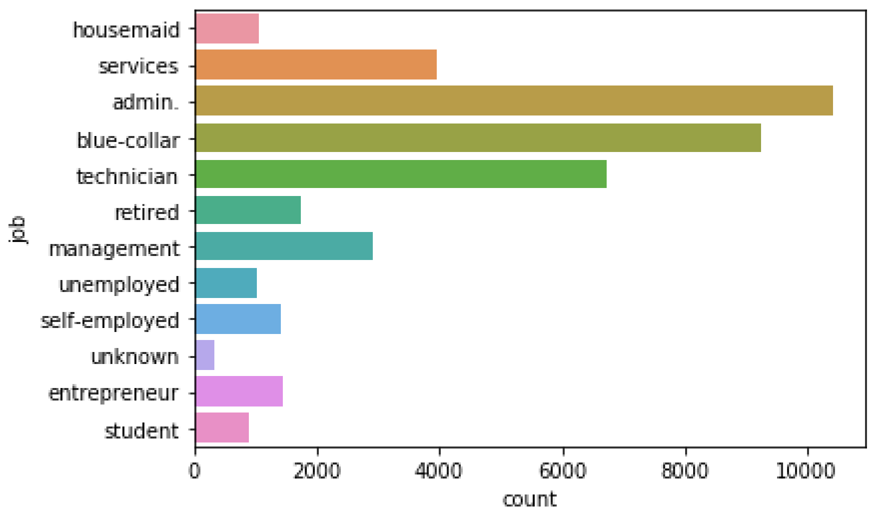
Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cc224a1588>
```



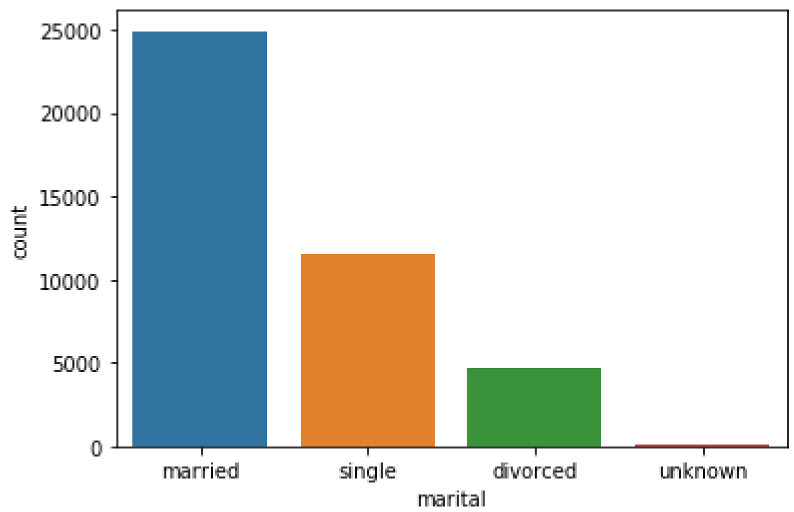
In [10]:
```
df[df['y']=='yes'].count()
#df['y'].count()
```

Out[10]:
```
age               4640
job               4640
marital           4640
education         4640
default           4640
housing           4640
loan              4640
campaign          4640
previous          4640
poutcome          4640
emp.var.rate      4640
cons.price.idx    4640
cons.conf.idx     4640
euribor3m         4640
nr.employed       4640
y                 4640
dtype: int64
```

In [11]:
```
sns.countplot(y='job', data=df)
```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1cc227c1988>

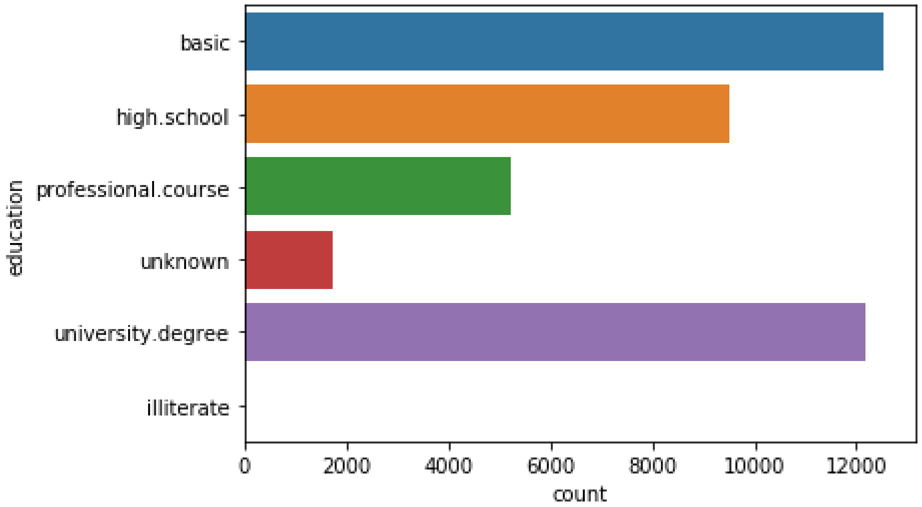


In [12]:
```
sns.countplot(x='marital', data=df)
```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1cc2285fac8>



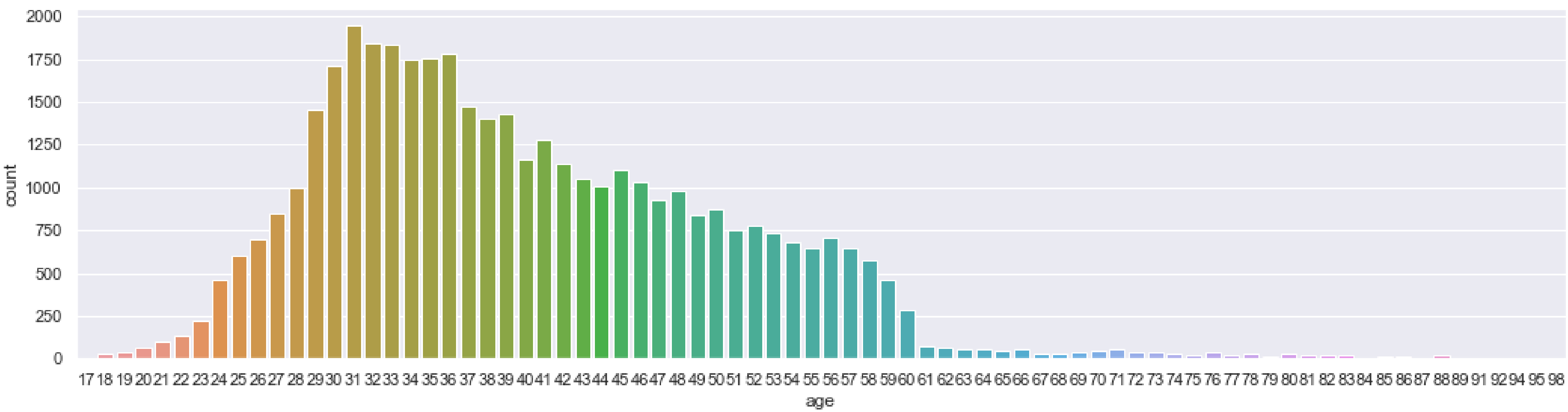
In [13]:
```
sns.countplot(y='education', data=df)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1cc228d4888>



In [14]:
```
sns.set(rc={'figure.figsize':(17.7,4.27)})
sns.countplot(x='age', data=df)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1cc2294a8c8>



2.3 Transforming the Data (Pre Processing)

Sklearn provides a very efficient tool for *encoding the levels of a categorical features into numeric values.* `LabelEncoder` encode labels with value between 0 and n_classes-1

In [15]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
```

In [16]:
```
le = preprocessing.LabelEncoder()
```

In [17]:
```
df.head(3)
```

Out[17]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
0	56	housemaid	married	basic	no	no	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
1	57	services	married	high.school	unknown	no	no	1	0	nonexistent	1.1	93.994	-36.4	4.857
2	37	services	married	high.school	no	yes	no	1	0	nonexistent	1.1	93.994	-36.4	4.857

In [18]:
```
df.job = le.fit_transform(df.job)  #fit_transform() is sub-function of LabelEncoder.
```

In [19]:
```
df.marital = le.fit_transform(df.marital)
```

In [20]:
```
df.education = le.fit_transform(df.education)
df.default = le.fit_transform(df.default)
df.housing = le.fit_transform(df.housing)
df.loan = le.fit_transform(df.loan)
df.poutcome = le.fit_transform(df.poutcome)
```

In [21]:
```
df.head()
```

Out[21]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employ
0	56	3	1	0	0	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
1	57	7	1	1	1	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
2	37	7	1	1	0	2	0	1	0	1	1.1	93.994	-36.4	4.857	519
3	40	0	1	0	0	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
4	56	7	1	1	0	0	2	1	0	1	1.1	93.994	-36.4	4.857	519

In [22]:
```
df.shape
```

Out[22]: (41188, 16)

3. Analysis and Modeling

3.1 Using data for Classification Task

In [23]:
```
import sklearn
import pickle
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [24]:
```
X = df.iloc[:,0:15]
X.head()
```

Out[24]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employ
0	56	3	1	0	0	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
1	57	7	1	1	1	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
2	37	7	1	1	0	2	0	1	0	1	1.1	93.994	-36.4	4.857	519
3	40	0	1	0	0	0	0	1	0	1	1.1	93.994	-36.4	4.857	519
4	56	7	1	1	0	0	2	1	0	1	1.1	93.994	-36.4	4.857	519

In [25]:
```
y = df.iloc[:,15]
y.head()
```

Out[25]:
```
0    no
1    no
2    no
3    no
4    no
Name: y, dtype: object
```

In [26]:
```
# Split the Dataset into Training and Test Datasets in to 80% and 20%
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.2, random_state=0)
```

In [27]:
```
x_train.shape, y_train.shape
```

Out[27]:
```
((32950, 15), (32950,))
```

In [28]:
```
x_test.shape, y_test.shape
```

Out[28]:
```
((8238, 15), (8238,))
```

In [29]:
```
y_train
```

Out[29]:
```
29321     no
23925     no
39148    yes
12078     no
41021     no
        ...
20757     no
32103     no
30403    yes
21243     no
2732      no
Name: y, Length: 32950, dtype: object
```

3.2 Modeling - Training the model

We will make use of 2 different classification algorithms (Logistic Regression and Decision Tree Classifier) to train this data set, record the accuracy on test set and compare it.

3.2.1 Decision Tree Classifier

In [30]:
```
model_dt = DecisionTreeClassifier(max_leaf_nodes=10, random_state=0)
model_dt.fit(x_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=10,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [35]:
```
prediction_dt = model_dt.predict(x_test)
```

In [38]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_true = y_test, y_pred = prediction_dt)
```

Out[38]:
```
0.8999757222626851
```

In [39]:
```
from sklearn.metrics import confusion_matrix  #call again in any time using
confusion_matrix = confusion_matrix(y_test, prediction_dt)
print(confusion_matrix)
```

```
[[7251   68]
 [ 756  163]]
```

3.2.2. Logistic Regression

In [40]:
```
model_lr=LogisticRegression(penalty='l2', max_iter=1000)
```

In [41]:
```
model_lr.fit(x_train, y_train)
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be cha
nged to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[41]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [42]:
```
prediction_lr=model_lr.predict(x_test)
```

In [43]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, prediction_lr)
```

Out[43]:
```
0.8999757222626851
```

In [44]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, prediction_lr)
print(confusion_matrix)
```

```
[[7260   59]
 [ 765  154]]
```

In [45]:
```
print(model_lr.coef_)
```

```
[[ 0.00590081  0.0097752   0.10272748  0.0554617  -0.16396652 -0.00278507
  -0.0134169  -0.02849879  0.21068573  0.64917634 -0.22683415  0.26672531
   0.02626022 -0.1532803  -0.00518502]]
```

From result, we can know that: most positive features are poutcome, previous, cons.price.idx. And most negative features are emp.var.rate, default, euribor3m.

In [46]:
```
df.describe()
```

Out[46]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.
count	41188.00000	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188
mean	40.02406	3.72458	1.172769	2.005608	0.208872	1.071720	0.327425	2.567593	0.172963	0.930101	0
std	10.42125	3.59456	0.608902	1.770171	0.406686	0.985314	0.723616	2.770014	0.494901	0.362886	1
min	17.00000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	-3
25%	32.00000	0.00000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	-1
50%	38.00000	2.00000	1.000000	1.000000	0.000000	2.000000	0.000000	2.000000	0.000000	1.000000	1
75%	47.00000	7.00000	2.000000	4.000000	0.000000	2.000000	0.000000	3.000000	0.000000	1.000000	1
max	98.00000	11.00000	3.000000	5.000000	2.000000	2.000000	2.000000	56.000000	7.000000	2.000000	1

In [47]:
```
X_axis = df['emp.var.rate']
Y_axis = df['previous']
```

In [48]:

```
Y_axis
```

Out[48]:

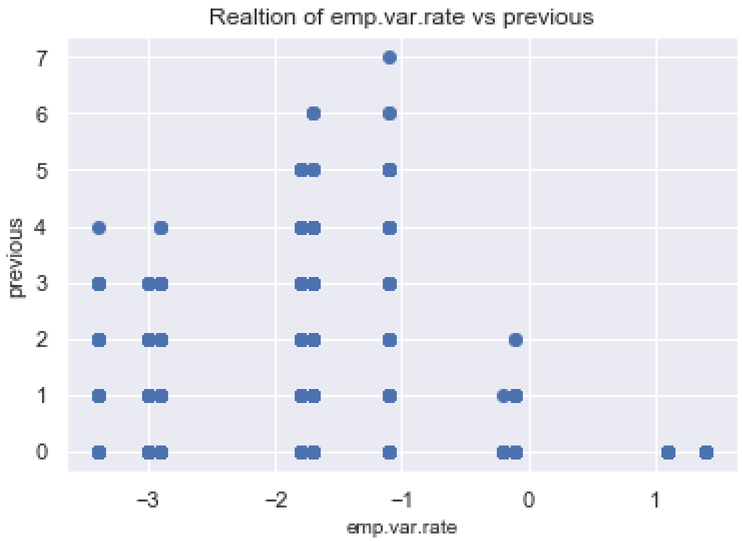
```
0        0
1        0
2        0
3        0
4        0
        ..
41183    0
41184    0
41185    0
41186    0
41187    1
Name: previous, Length: 41188, dtype: int64
```

In [49]:
```
# To make a scatter plot for 2 features: previous, emp.var.rate
%matplotlib inline
import matplotlib.pyplot as plt

fig, axes = plt.subplots()

# Xlabel, Ylabel, Xticks, Grid lines
axes.set_xlabel('emp.var.rate', fontsize=10)
axes.set_ylabel('previous', fontsize=11)
axes.set_title('Realtion of emp.var.rate vs previous', fontsize=12)
axes.yaxis.grid(True)

# Call print scatter function and show function.
plt.scatter(X_axis,Y_axis)
plt.show()
```



3.2.3. k-Means Clustering

Based on scatter plot, we can choose n_clusters=4 for the model.

In [50]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from itertools import cycle, islice
from pandas.plotting import parallel_coordinates
```

In [51]:
```
# To keep values of different columns comparable, we scale the values in these features, by StandardScaler
copy_X = X.copy()
scale_X = StandardScaler().fit_transform(copy_X)  #we use the fit_transform function of it, and we give the select_df
scale_X
```

Out[51]:
```
array([[ 1.53303429, -0.20157925, -0.2837415 , ...,  0.88644656,
         0.71245988,  0.33167991],
       [ 1.62899323,  0.91122681, -0.2837415 , ...,  0.88644656,
         0.71245988,  0.33167991],
       [-0.29018564,  0.91122681, -0.2837415 , ...,  0.88644656,
         0.71245988,  0.33167991],
       ...,
       [ 1.53303429,  0.35482378, -0.2837415 , ..., -2.22495344,
        -1.49518647, -2.8156966 ],
       [ 0.38152696,  1.46762984, -0.2837415 , ..., -2.22495344,
        -1.49518647, -2.8156966 ],
       [ 3.26029527,  0.35482378, -0.2837415 , ..., -2.22495344,
        -1.49518647, -2.8156966 ]])
```

In [52]:
```
kmeans = KMeans(n_clusters=4)
model_km = kmeans.fit(scale_X)
print("model\n", model_km)
```

```
model
 KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [53]:
```
centers = model_km.cluster_centers_
centers
```

Out[53]:
```
array([[ 3.44008368e-01, -3.54028206e-02, -1.63227638e-01,
        -2.75684487e-01,  1.94636572e+00, -5.54933028e-02,
        -7.24226187e-04,  1.08410469e-01, -3.47502113e-01,
         1.94562720e-01,  7.27357594e-01,  6.20139188e-01,
         2.82848960e-01,  7.25476563e-01,  6.46354592e-01],
       [-1.03701761e-01,  5.93860032e-03, -2.93488336e-02,
         7.48482236e-02, -5.13599691e-01, -3.75198239e-02,
        -5.23200813e-04,  8.68943369e-02, -3.46344036e-01,
         1.96641177e-01,  6.60948986e-01,  4.44185992e-01,
         2.23855332e-01,  6.91129932e-01,  6.48516661e-01],
       [-3.71113391e-03,  8.92721163e-03,  5.56940347e-02,
         7.44610179e-03, -2.18140658e-01,  6.32858748e-02,
         4.95879842e-03, -2.04573557e-01,  2.10376000e+00,
        -2.48639035e+00, -1.12342632e+00, -8.51360253e-01,
        -4.74826023e-01, -1.14282623e+00, -1.05143489e+00],
       [-3.68396753e-02,  9.58169295e-03,  1.51864091e-01,
         4.41703868e-02, -2.69521479e-01,  8.76600852e-02,
        -5.89512694e-04, -1.63307866e-01,  2.87739255e-02,
         5.48612183e-01, -1.36090748e+00, -9.66842055e-01,
        -4.46011528e-01, -1.41176461e+00, -1.30915668e+00]])
```

Plotting clusters/groups

After find out cluster centers, next we plot them for clear visualization.

In [54]:
```
# Function that creates a DataFrame with a column for Cluster Number
def pd_centers(featuresUsed, centers):
        colNames = list(featuresUsed)
        colNames.append('prediction')

        # Zip with a column called 'prediction' (index)
        Z = [np.append(A, index) for index, A in enumerate(centers)]

        # Convert to pandas data frame for plotting
        P = pd.DataFrame(Z, columns=colNames)
        P['prediction'] = P['prediction'].astype(int)
        return P
```

In [55]:

```
# Function that creates Parallel Plots
def parallel_plot(data):
        my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(data)))
        plt.figure(figsize=(15,8)).gca().axes.set_ylim([-3,+3])
        parallel_coordinates(data, 'prediction', color = my_colors, marker='o')
```

In [56]:

```
features = ['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'campaign', 'previous', 'poutcome',
            'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']

P = pd_centers(features, centers)
P
```

Out[56]:

	age	job	marital	education	default	housing	loan	campaign	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.id
0	0.344008	-0.035403	-0.163228	-0.275684	1.946366	-0.055493	-0.000724	0.108410	-0.347502	0.194563	0.727358	0.620139	0.28284
1	-0.103702	0.005939	-0.029349	0.074848	-0.513600	-0.037520	-0.000523	0.086894	-0.346344	0.196641	0.660949	0.444186	0.22385
2	-0.003711	0.008927	0.055694	0.007446	-0.218141	0.063286	0.004959	-0.204574	2.103760	-2.486390	-1.123426	-0.851360	-0.47482
3	-0.036840	0.009582	0.151864	0.044170	-0.269521	0.087660	-0.000590	-0.163308	0.028774	0.548612	-1.360907	-0.966842	-0.44601

```
In [57]: parallel_plot(P)
```